Combinational Circuit Design

- Part I: Design Procedure and Examples
- Part II : Arithmetic Circuits
- Part III : Multiplexer, Decoder, Encoder, Hamming Code

Combinational Circuits

A combinational circuit has:

- *n* Boolean inputs (1 or more),
- *m* Boolean outputs (1 or more)
- logic gates mapping the inputs to the outputs

Design Procedure

1. Specification
   - Write a complete specification for the circuit
   - Specify/Label input and output
2. Formulation
   - Derive a truth table or initial Boolean equations that define the required relationships between the inputs and outputs, if not in the specification
   - Apply hierarchical design if appropriate
3. Optimization
   - Apply 2-level and multiple-level optimization (Boolean Algebra, K-Map, software)
   - Draw a logic diagram for the resulting circuit using necessary logic gates.

Design Procedure (Cont.)

4. Technology Mapping
   - Map the logic diagram to the implementation technology selected (e.g. map into NANDs)
5. Verification
   - Verify the correctness of the final design manually or using a simulation tool

Practical Considerations:
- Cost of gates (Number)
- Maximum allowed delay
- Fan-in/Fan-out (# of Input ports/Output ports provided by devices)

Example 1

- **Question:** Design a circuit that has a 3-bit binary input and a single output (f) specified as follows:
  - F = 0, when the input is less than $(5)_{10}$
  - F = 1, otherwise
- **Solution**:
- Step 1 (Specification):
- Label the inputs (3 bits) as X, Y, Z
  - X is the most significant bit, Z is the least significant bit
- The output (1 bit) is F:
  - F = 1 → $(101)_2$, $(110)_2$, $(111)_2$
  - F = 0 → other inputs

Example 1 (cont.)

Step 2 (Formulation)

Obtain Truth table

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Boolean Expression:
$F = XY'Z+XYZ'+XYZ$

Step 3 (Optimization)

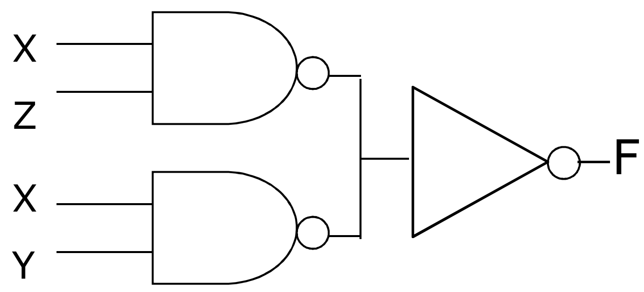
$F = XY'Z+XYZ'+XYZ$

$= XY'Z+XYZ'+XYZ+XZ+XY$

$= XZ + XY$

(Use consensus theorem)

Circuit Diagram



Example 2

- **Question (BCD to Excess-3 Code Converter)**

- Code converters convert from one code to another (BCD to Excess-3 in this example)
- The inputs are defined by the code that is to be converted (BCD in this example)
- The outputs are defined by the converted code (Excess-3 in this example)
- Excess-3 code is a decimal digit plus three converted into binary, i.e., 0 is 0011, 1 is 0100, etc.

Example 2 (cont.)

Step 1 (Specification)

- 4-bit BCD input (A,B,C,D)
- 4-bit E-3 output (W,X,Y,Z)

Step 2 (Formulation)

Obtain Truth table

	BCD Input				Excess 3 Output			
Decimal	**A**	**B**	**C**	**D**	**W**	**X**	**Y**	**Z**
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
10-15	All other inputs				X	X	X	X

Example 2 (cont.)

Step 3 (Optimization)

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

W = A + BC + BD

AB \ CD	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	X	X	X	X
10	0	1	X	X

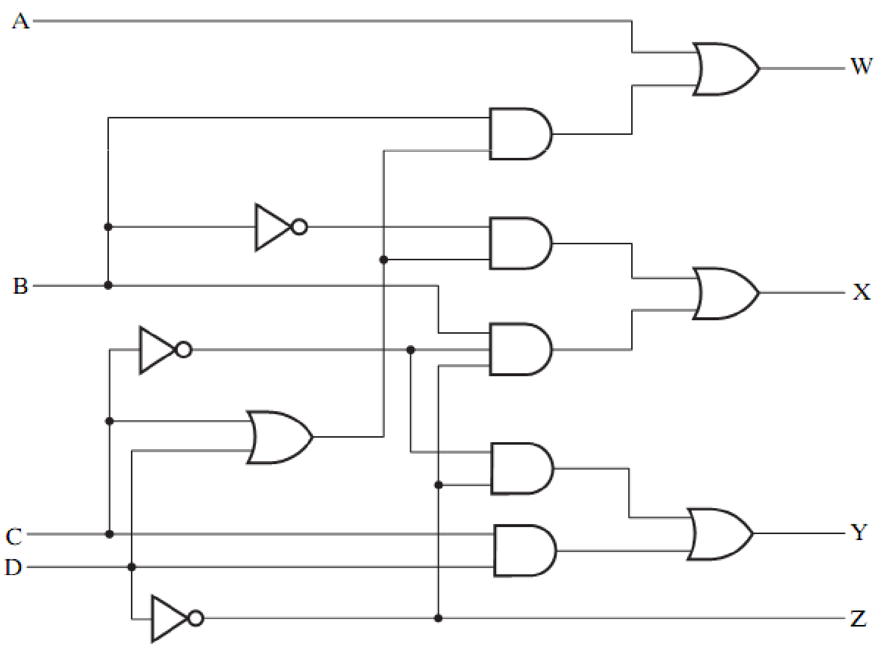
X = B'C + BC'D' + B'D

AB \ CD	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	X	X	X	X
10	1	0	X	X

Y = CD + C'D'

AB \ CD	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	X	X	X	X
10	1	0	X	X

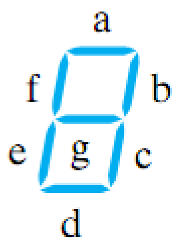
Z = D'



source: Mano's book

Example 3

Question (BCD-to-Seven-Segment Decoder)



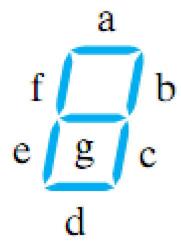
src: Mano's book

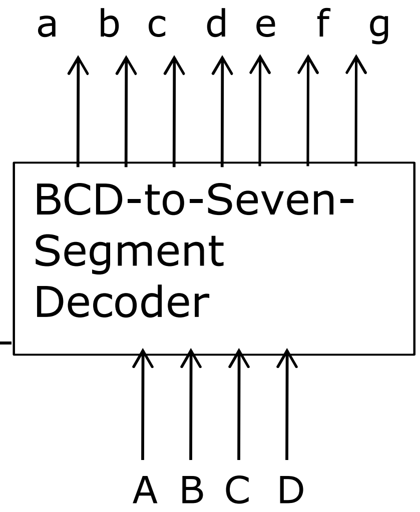
- A seven-segment display is digital readout found in electronic devices like clocks, TVs, etc.
  - Made of seven light-emitting diodes (LED) segments; each segment is controlled separately.
- **A BCD-to-Seven-Segment decoder** is a combinational circuit
  - Accepts a decimal digit in BCD (input)
  - Generates appropriate outputs for the segments to display the input decimal digit (output)

Example 3 (cont.)

Step 1 (Specification):

- 4 inputs (A, B, C, D)
- 7 outputs (a, b, c, d, e, f, g)





Step 2 (Formulation)

Decimal	BCD Input				7 Segment Decoder						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
10-15	All Other Inputs				0	0	0	0	0	0	0

Invalid BCD codes = No Light

Example 3 (cont.)

Step 3 (Optimization)

a

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	0	0	0
10	1	1	0	0

b

AB \ CD	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	0	0	0	0
10	1	1	0	0

c

AB \ CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	0	0	0	0
10	1	1	0	0

d

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	0	0	0	0
10	1	1	0	0

e

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	0	0	0	0
10	1	0	0	0

f

AB \ CD	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	0	0	0	0
10	1	1	0	0

g

AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	0	0	0
10	1	1	0	0

Example 3 (cont.)

Step 3 (Optimization) (cont.)

$a = A'C + A'BD + AB'C' + B'C'D'$

$b = A'B' + A'C'D' + A'CD + B'C'$

$c = A'B + B'C' + A'C' + A'D$

$d = A'CD' + A'B'C + B'C'D' + AB'C' + A'BC'D$

$e = A'CD' + B'C'D'$

$f = A'BC' + A'C'D' + A'BD' + AB'C'$

$g = A'CD' + A'B'C + A'BC' + AB'C'$

Exercise: Draw the circuit

Part II Arithmetic Circuits

- Adder
- Subtractor
- Carry Look Ahead Adder
- BCD Adder
- Multiplier

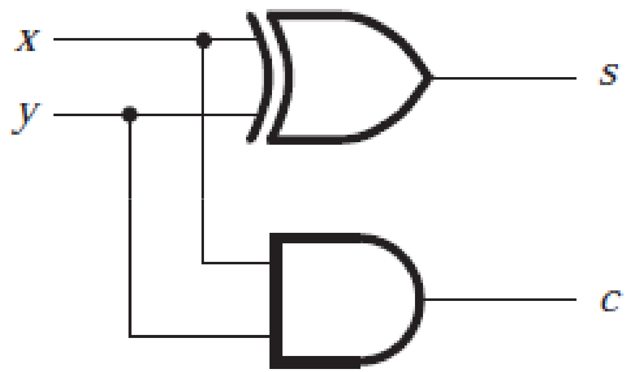
Half Adder

Design a half-Adder for 1-bit numbers

- **1. Specification:**
  Optimization/Circuit
  2 inputs (X,Y)
  2 outputs (C,S)
- **2. Formulation:**

x	*y*	*c*	*s*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

3. Logic Diagram



Graphical Symbol

Full Adder

A combinational circuit that adds 3 input bits (x_i, y_i, c_{in}) to generate a Sum bit and a Carry-out bit

c_i	x_i	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$s_i = (\overline{x}_i y_i + x_i \overline{y}_i)\overline{c}_i + (\overline{x}_i \overline{y}_i + x_i y_i)c_i$$

$$= (x_i \oplus y_i)\overline{c}_i + \overline{(x_i \oplus y_i)}c_i$$

$$= (x_i \oplus y_i) \oplus c_i$$

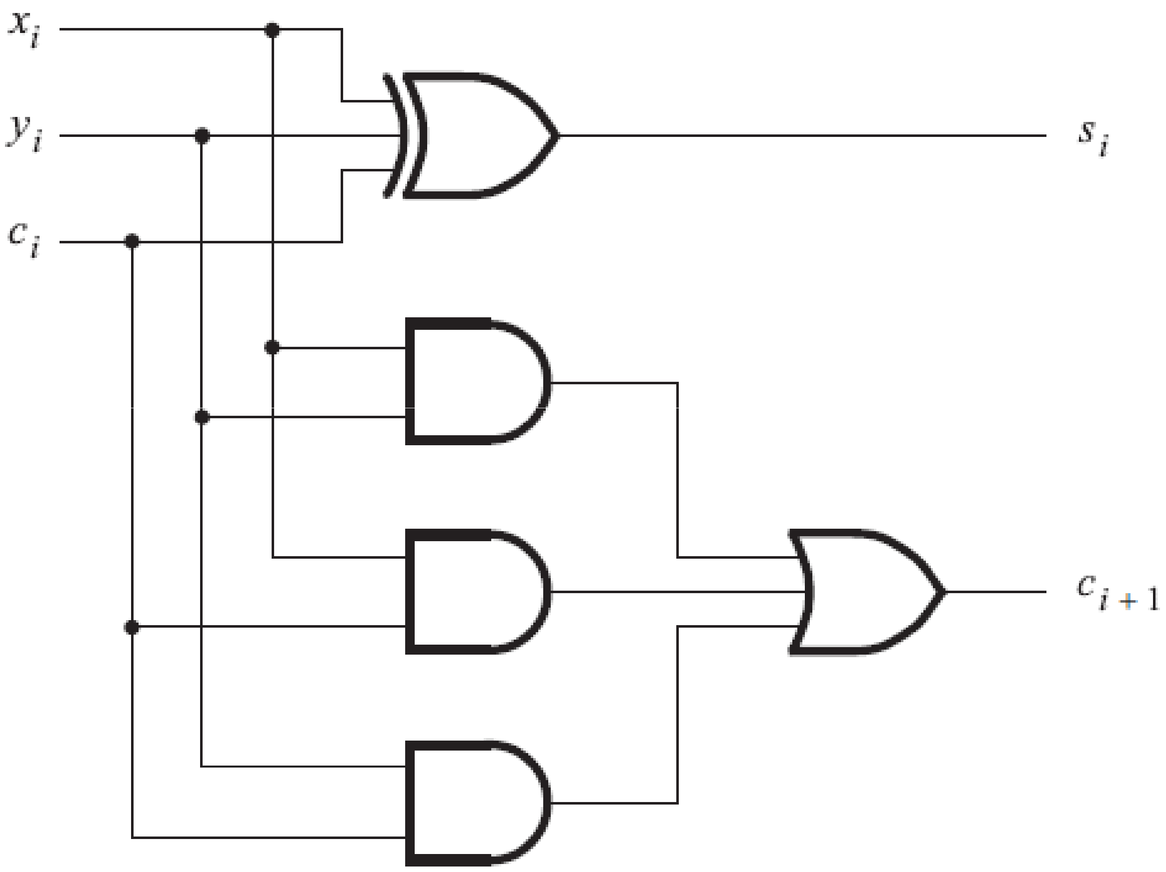
c_i \ $x_i y_i$	00	01	11	10
0		1		1
1	1		1	

$$s_i = x_i \oplus y_i \oplus c_i$$

c_i \ $x_i y_i$	00	01	11	10
0			1	
1		1	1	1

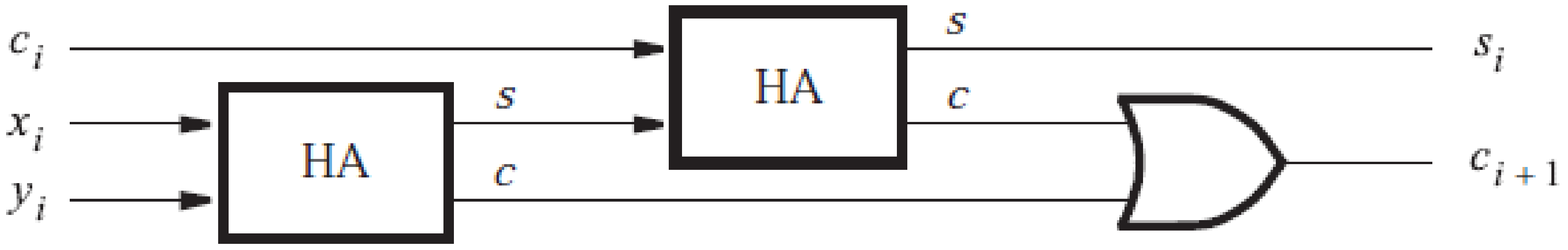
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

Full Adder Logic Diagram

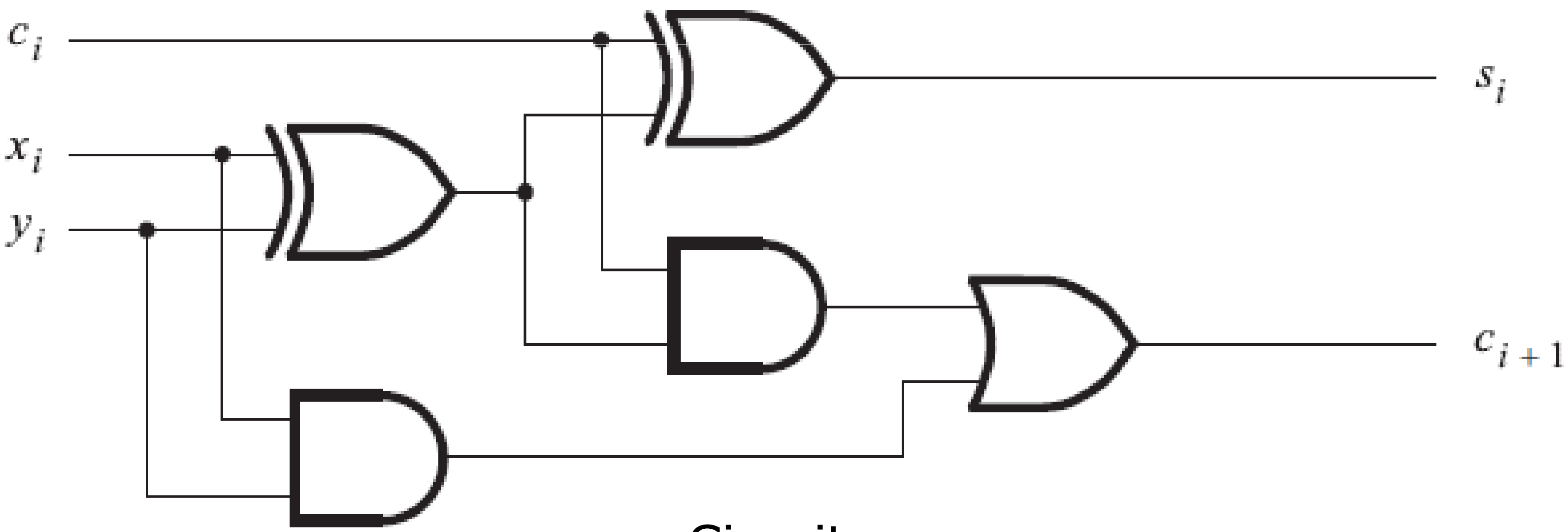


From Brown's Fundamentals of digital logic

Full Adder = 2 Half Adders



Block diagram



Circuit

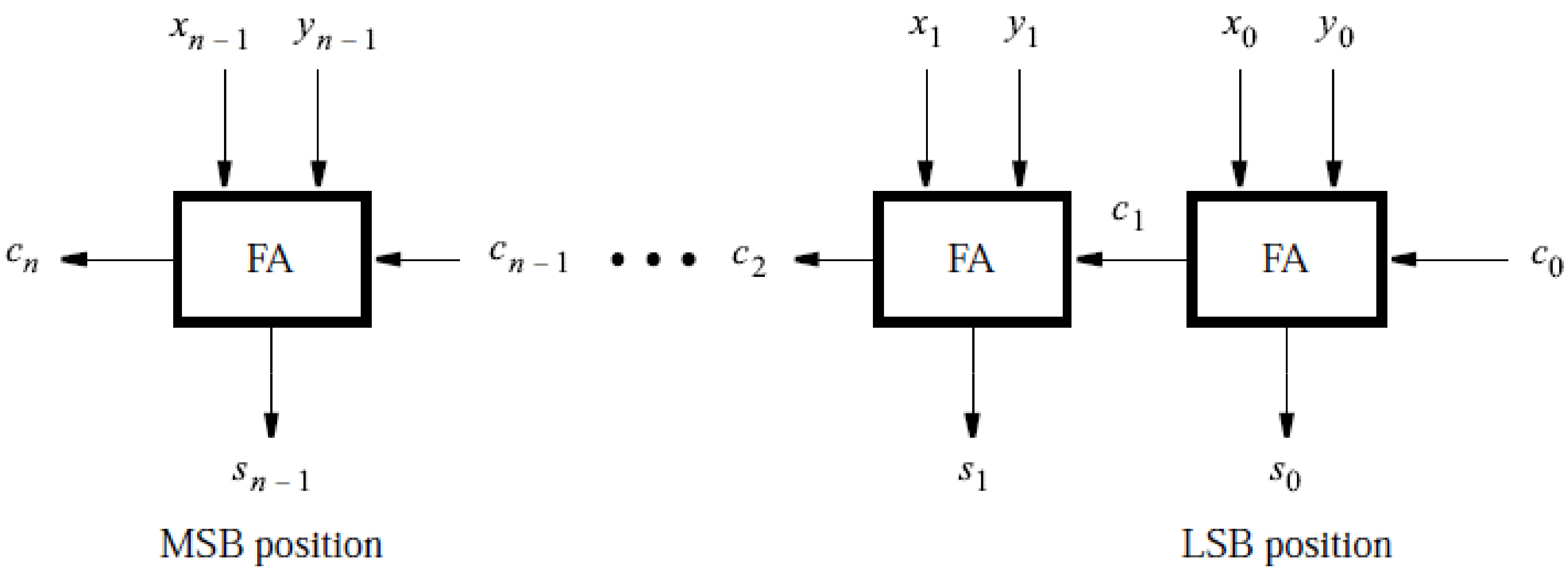
Exercise : Verify this full-adder implementation.

From Brown's Fundamentals of digital logic

Bigger Adders

- How to build an adder for n-bit numbers?
  - Example: 4-Bit Adder
    - Inputs ? 9 inputs
    - Outputs ? 5 outputs
    - What is the size of the truth table? 512 rows!
    - How many functions to optimize? 5 functions

Ripple Carry Adder

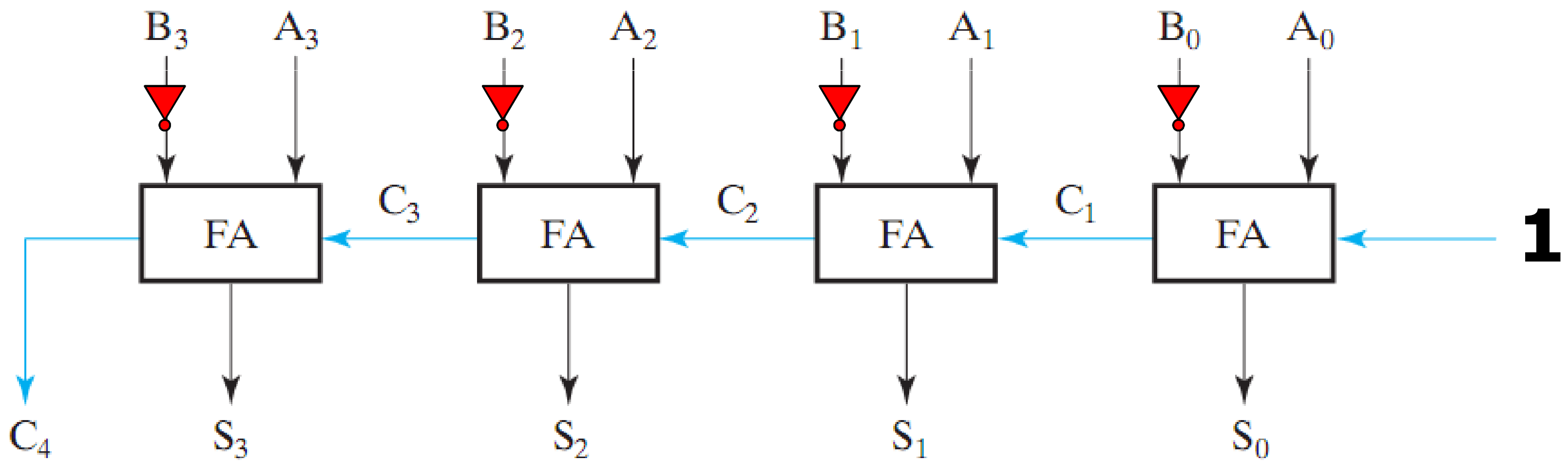


Note:

- Carry signal "ripples" through the full-adder stages.
- Delay can be an issue.

Subtraction (2's Complement)

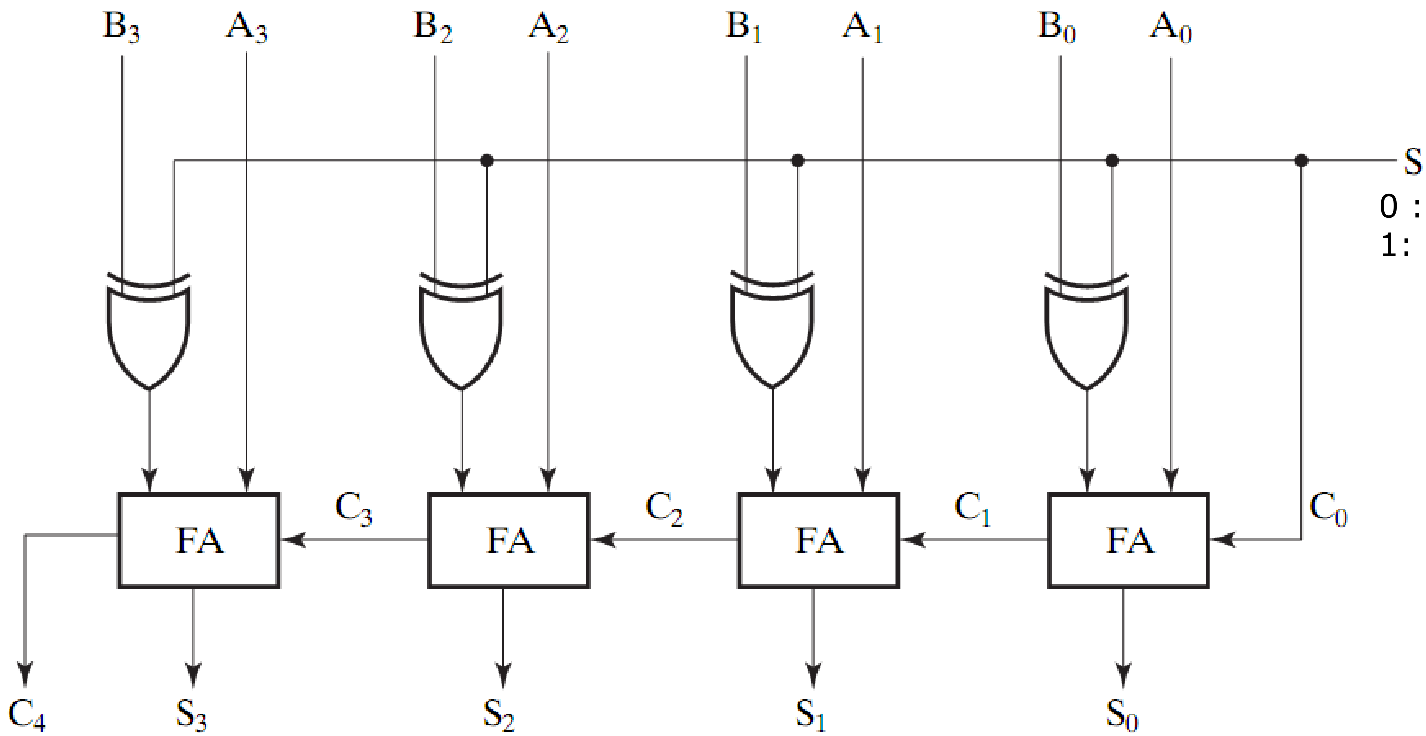
- How to build a subtractor using 2's complement?



Src: Mano's Book

S = A + (-B)

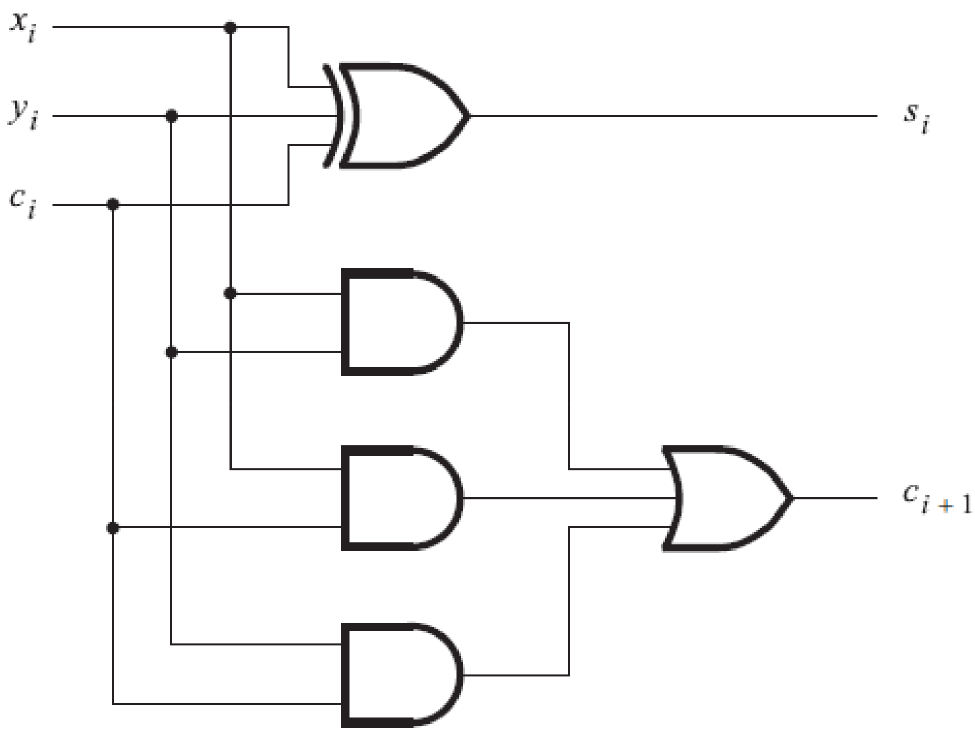
Adder/Subtractor



Src: Mano's Book

Using full adders and XOR we can build an Adder/Subtractor!

Full-Adder (Review)



$$s_i = x_i \oplus y_i \oplus c_i; c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

Carry-Lookahead Adder (CLA)

Define $g_i = x_i y_i ; p_i = x_i + y_i$

Then $c_{i+1} = g_i + p_i c_i$

- g_i is called "*generate*" function and p_i is called "*propagate*" function.
- Rewriting c_{i+1} in terms of i-1 terms yields

$$c_{i+1} = g_i + p_i (g_{i-1} + p_{i-1} c_{i-1})$$

$$= g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1}$$

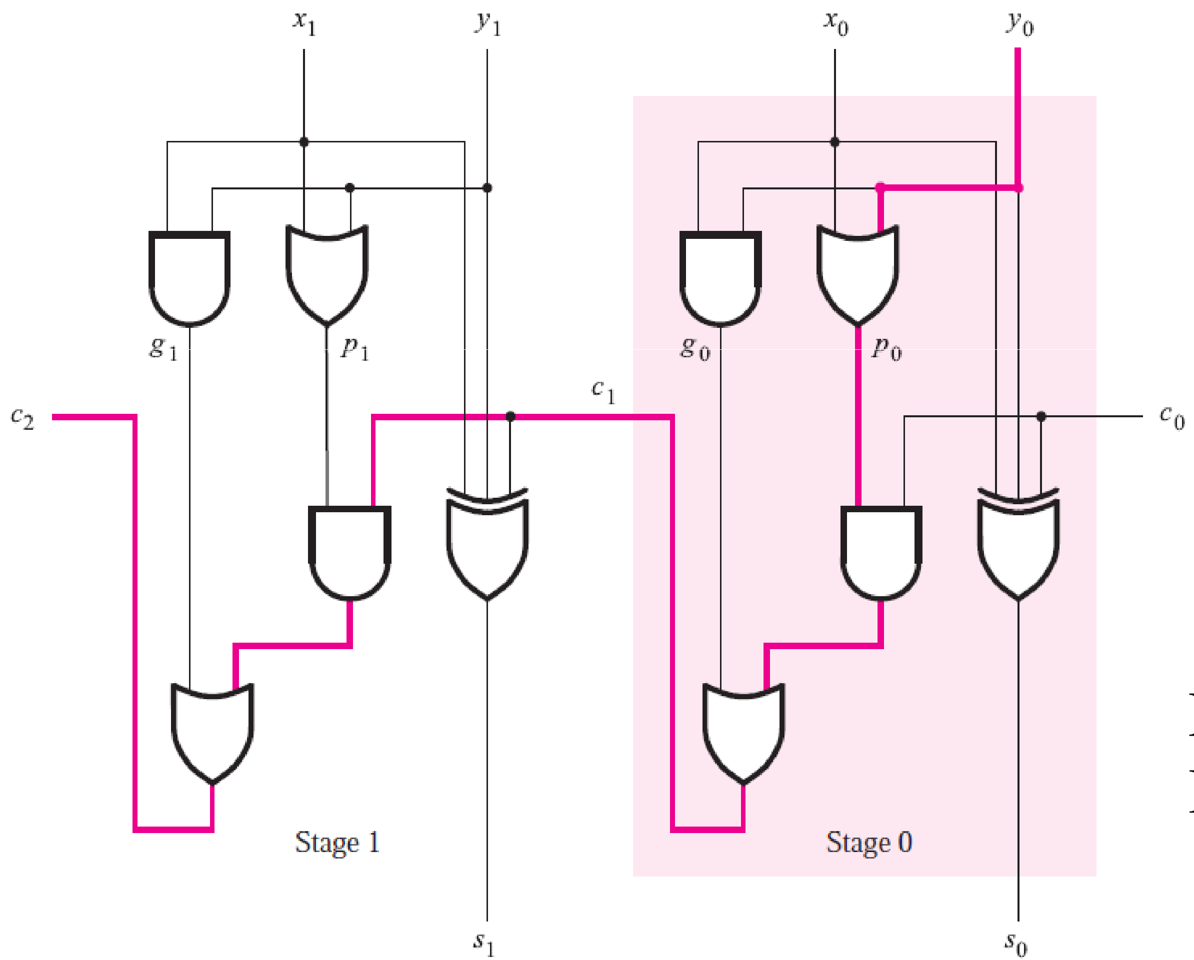
CLA (cont.)

- Repeating until 0 term yields

$$c_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \cdots + p_i p_{i-1} \cdots p_2 p_1 g_0$$
$$+ p_i p_{i-1} \cdots p_2 p_1 p_0 c_0$$

- c_{i+1} can be implemented in 2-level AND-OR circuits.
- A Carry-Lookahead Adder is based on this expression.

Ripple-carry Adder Delay



Only First 2 stages shown

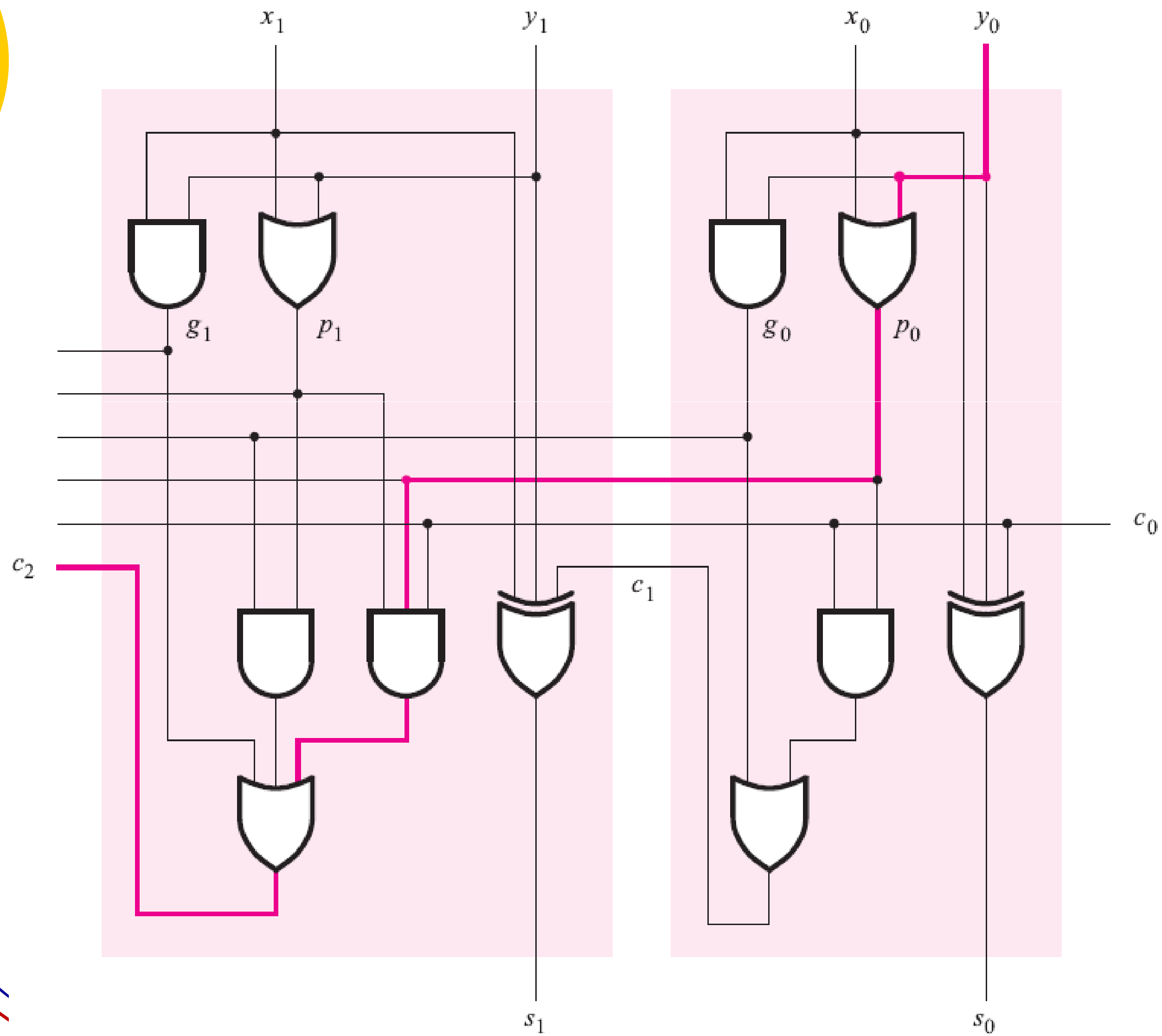
LSB:
$(x_0 , y_0) = (0,1)$

Delay: 5 gates

For n stages:
Delay: 2n+1 gates

From Brown's Fundamentals of digital logic

CLA Delay



Only First 2 stages shown

LSB:

$(x_0, y_0) = (0,1)$

$$c_1 = g_0 + p_0 c_0$$

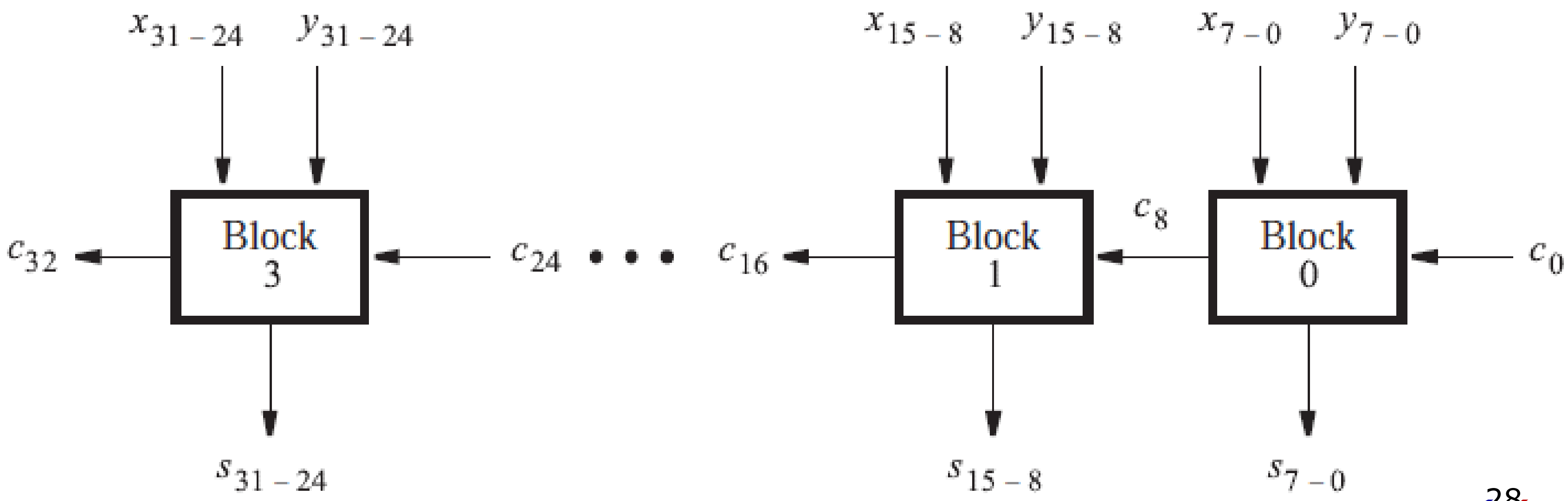
$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

Delay: 3 gates

For n stages:
Delay: 3 gates

CLA Implementation

- Total delay : 4 gates (1 for all g_i ,p_i , 2 for all carry, 1 for the final XOR to compute all s_i)
- Becomes very complex when n large.
- Hierarchical CLA with ripple-carry



CLA : A better implementation

Consider c_8 out of block 0:

$$c_8 = g_7 + p_7 g_6 + p_7 p_6 g_5 + \cdots + p_7 p_6 \cdots p_2 p_1 g_0$$

$$+ p_7 p_6 \cdots p_2 p_1 p_0 c_0$$

Recall that $c_1 = g_0 + p_0 c_0$

If define $P_0 = p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0 c_0$

$$G_0 = g_7 + p_7 g_6 + p_7 p_6 g_5 + \cdots + p_7 p_6 \cdots p_2 p_1 g_0$$

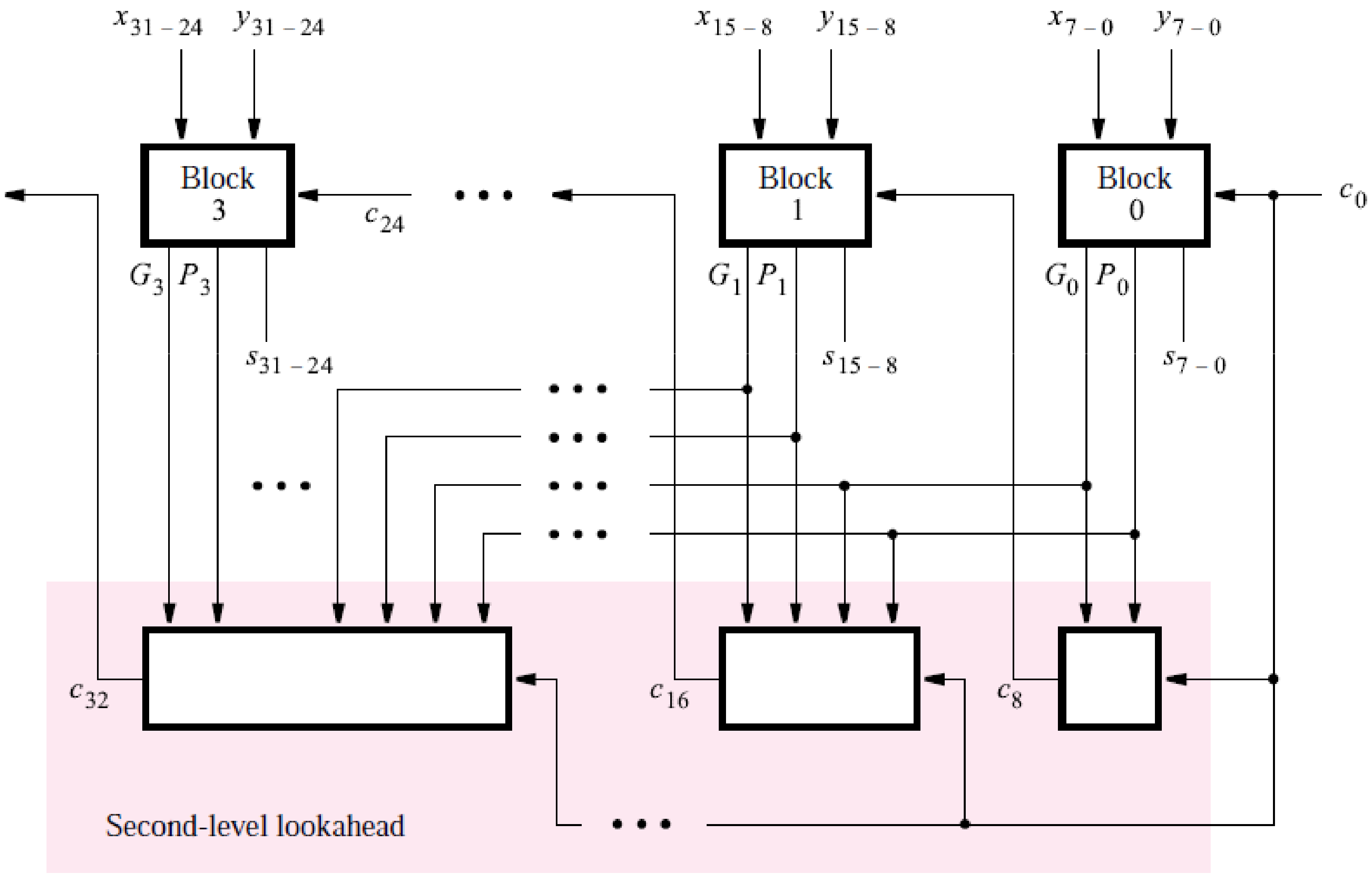
Then can write $c_8 = G_0 + P_0 c_0$

Likewise $c_{16} = G_1 + P_1 G_0 + P_1 P_0 c_0$

$$c_{16} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$$

$$c_{32} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0$$

CLA : A better implementation



BCD Addition

Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

$$Z = X + Y$$

If $Z \leq 9$, then $S = Z$ and carry-out $= 0$

if $Z > 9$, then $S = Z + 6$ and carry-out $= 1$

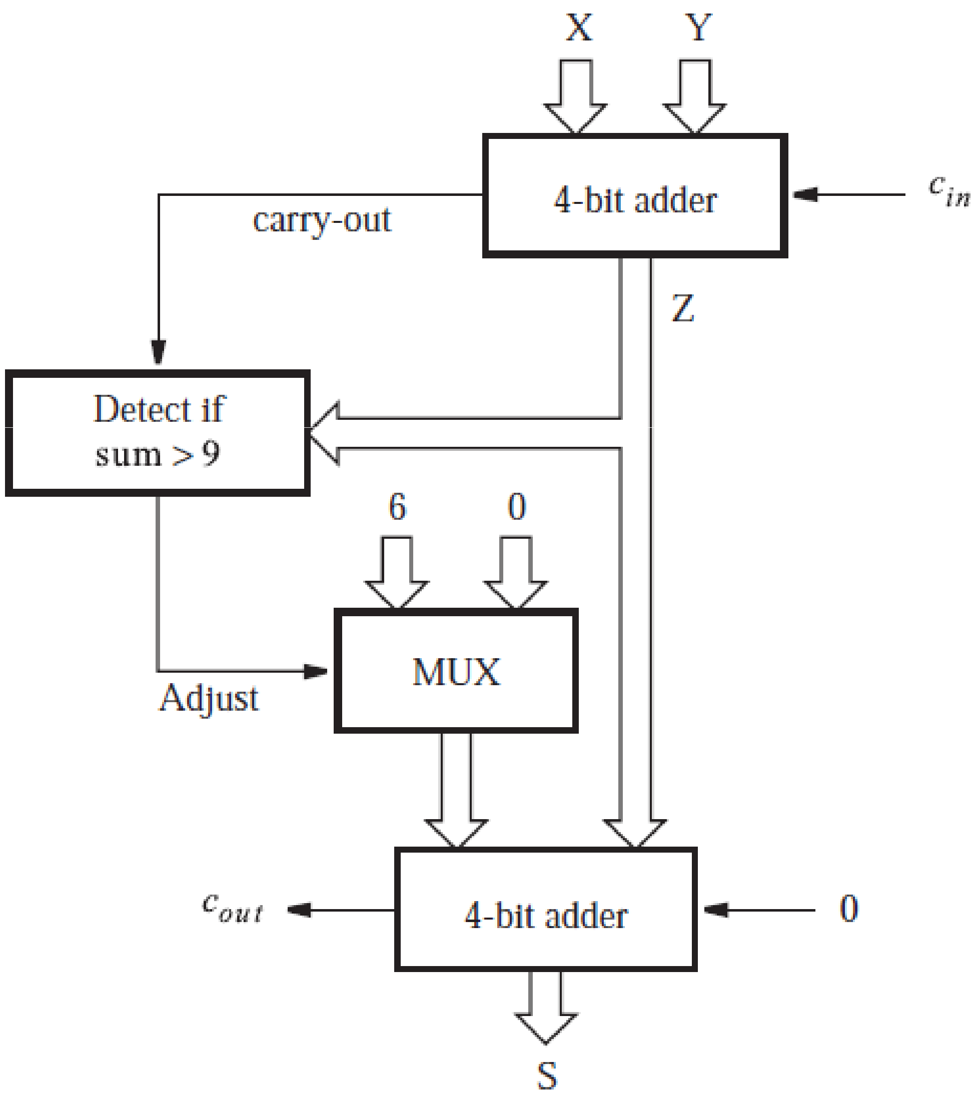
$$\begin{array}{r} X \\ +\ Y \\ \hline Z \end{array} \qquad \begin{array}{r} 0111 \\ +\ 0101 \\ \hline 1100 \\ +\ 0110 \\ \hline 10010 \end{array} \qquad \begin{array}{r} 7 \\ +\ 5 \\ \hline 12 \end{array}$$

carry → 1 0010, $S = 2$

$$\begin{array}{r} X \\ +\ Y \\ \hline Z \end{array} \qquad \begin{array}{r} 1000 \\ +\ 1001 \\ \hline 10001 \\ +\ 0110 \\ \hline 10111 \end{array} \qquad \begin{array}{r} 8 \\ +\ 9 \\ \hline 17 \end{array}$$

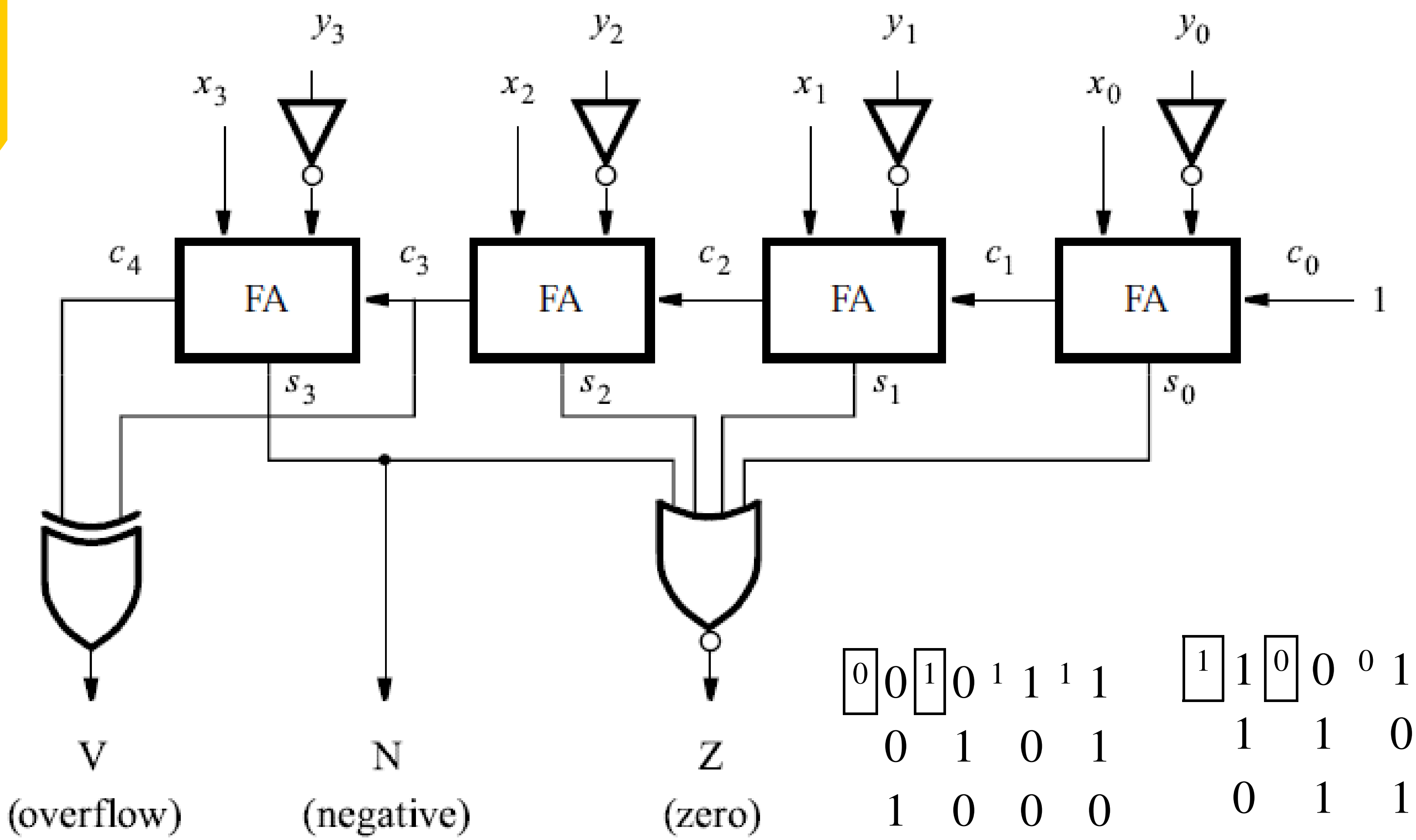
carry → 1 0111, $S = 7$

BCD Adder



Adjust=0 -> $S = Z + 0$
Adjust=1 -> $S = Z + 6$

4-bit Comparator



[0]	0	[1]	0	1	1	1	1
	0		1		0		1
	1		0		0		0

3-(-5)=-8

[1]	1	[0]	0	0	1	0	1
	1		1		0		0
	0		1		1		1

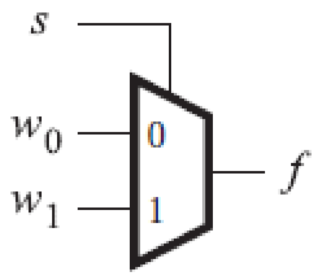
-5-4=7

4-bit Comparator

- $X < Y$
  - Same sign: No overflow (V=0) and N=1
  - Different sign: V=0 && N=1, OR V=1 (overflow) && N=0 (positive)
  - Thus, condition is $N \oplus V = 1$.
- $X = Y$ -> Z = 1
- $X > Y$
  - Same sign: No overflow (V=0) and N=0
  - Different sign: V=0 && N=0, OR V=1 (overflow) && N=1 (negative)
  - Thus, condition is $N \oplus V = 0$, i.e., the complement of $N \oplus V$, $(N \oplus V)'$.

2-to-1 Multiplexer (MUX)

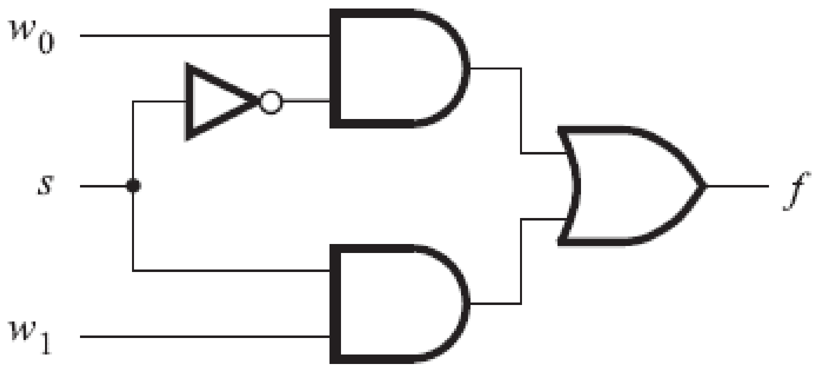
Multiplexer has multiple inputs and one output; it passes the signal on one input to the output.



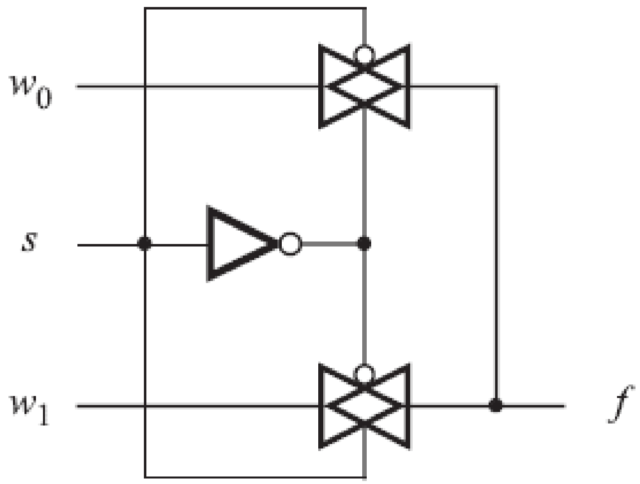
Symbol

s	f
0	w_0
1	w_1

Truth Table



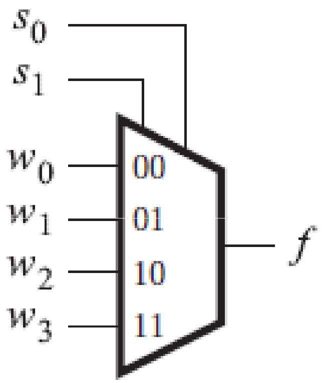
SOP circuit



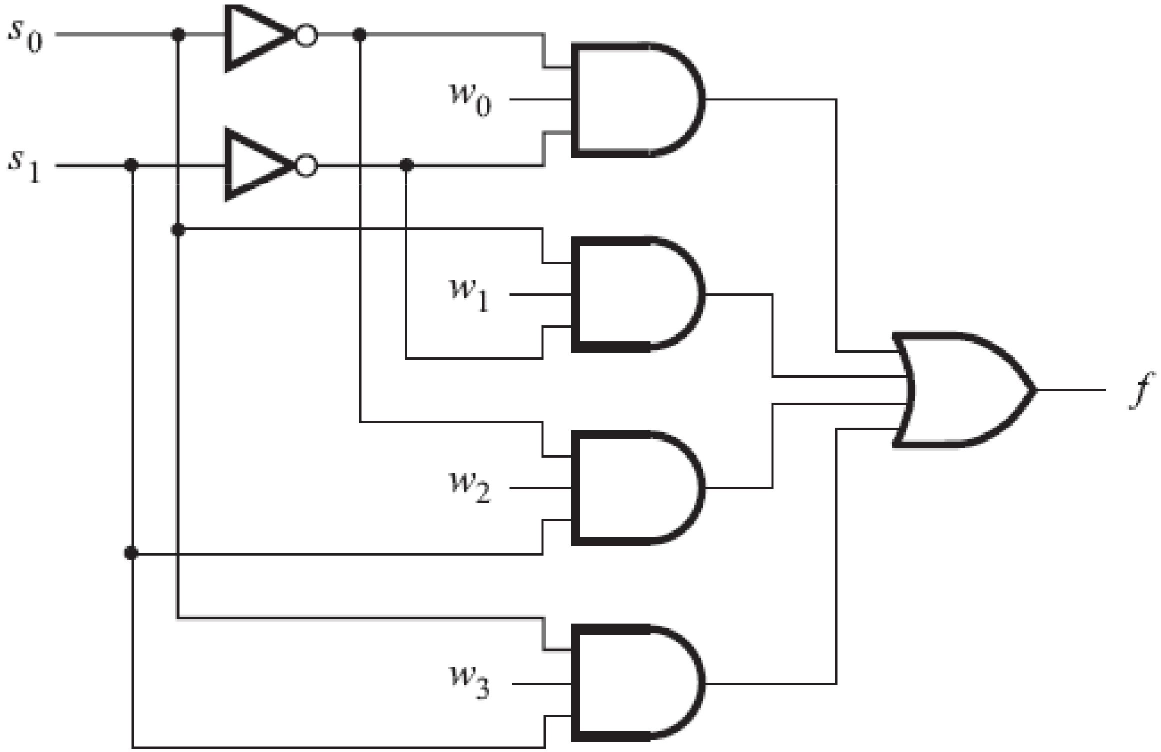
Circuit with transmission gates

4-to-1 Multiplexer

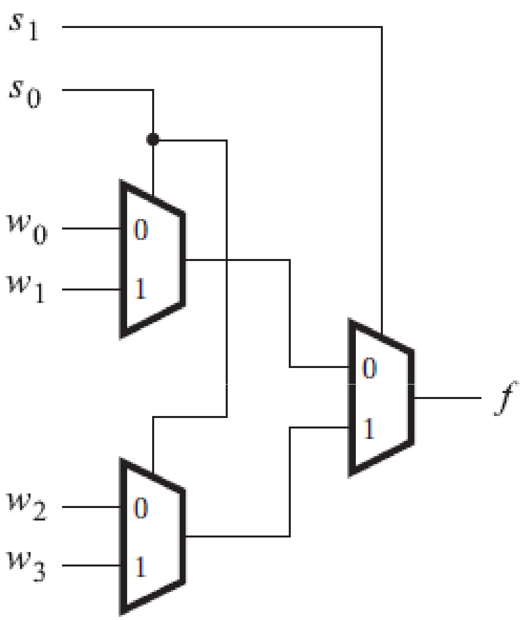
$$f = s_1's_0'w_0 + s_1's_0w_1 + s_1s_0'w_2 + s_1s_0w_3$$



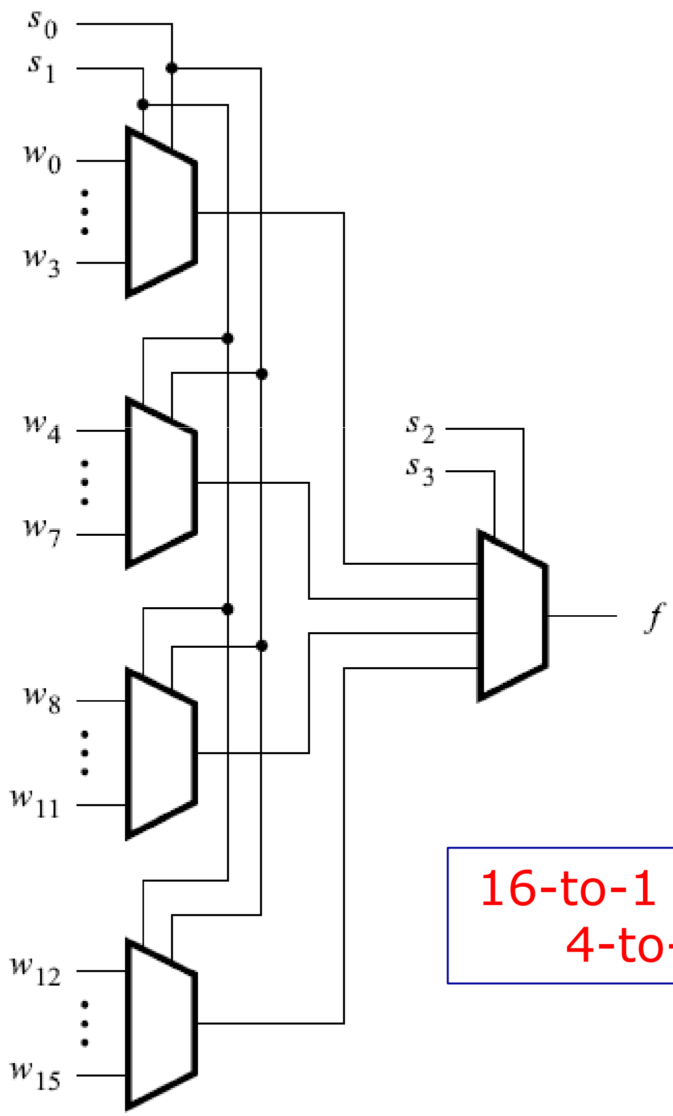
s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3



4-to-1 Multiplexer



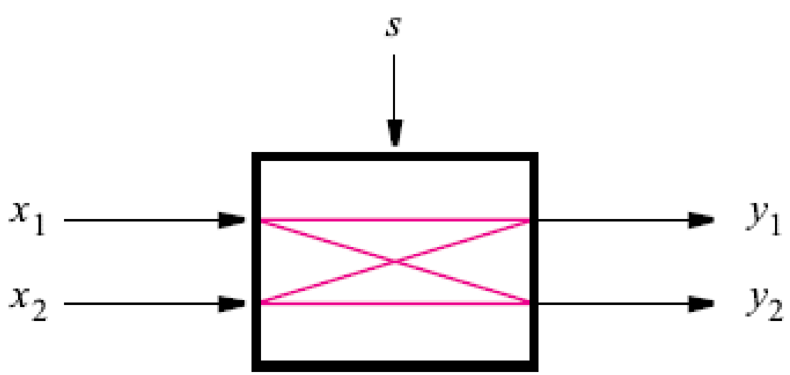
4-to-1 mux using
2-to-1 mux

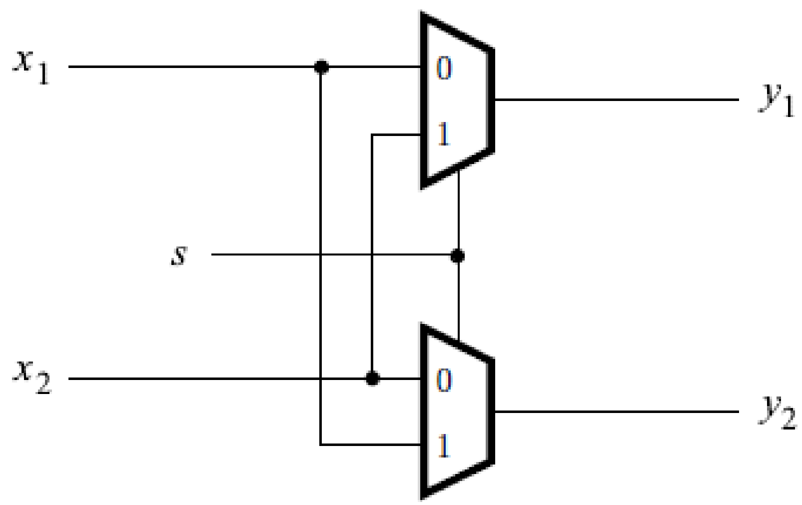


16-to-1 mux using
4-to-1 mux

2×2 crossbar switch

- 2 inputs, 2 outputs
- s = 0 -> connect x_1->y_1, x_2->y_2
- s = 1 -> connect x_1->y_2, x_2->y_1

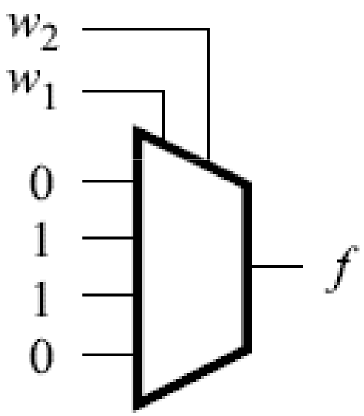




Synthesis of Logic Functions

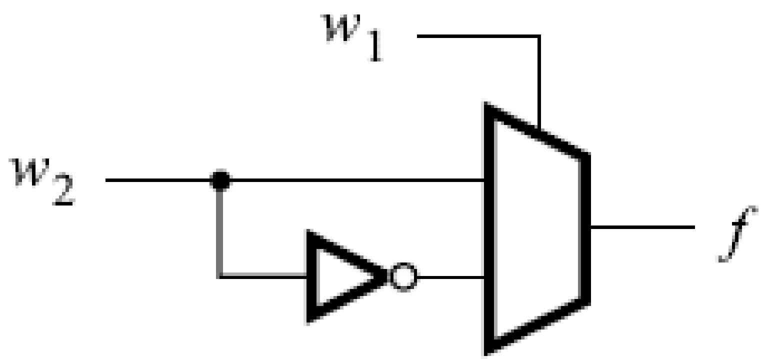
$$f = w_1 \oplus w_2$$

w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0



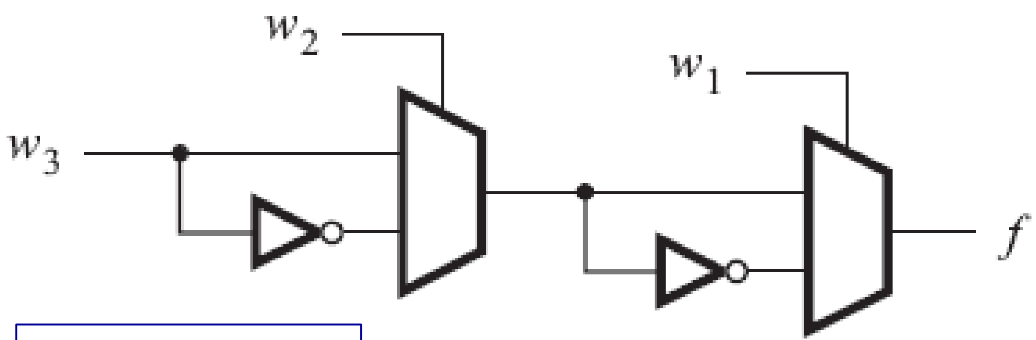
w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0

w_1	f
0	w_2
1	$\overline{w_2}$

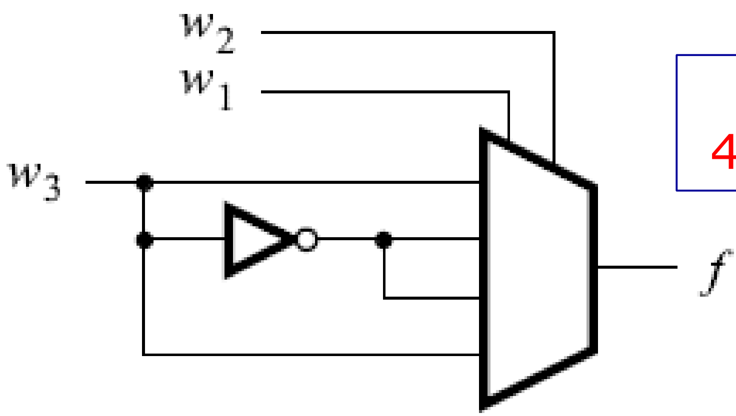


3-input XOR

w_1	w_2	w_3	f	
0	0	0	0	$w_2 \oplus w_3$
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	1	$\overline{w_2 \oplus w_3}$
1	0	1	0	
1	1	0	0	
1	1	1	1	



Using
2-to-1 MUX



Using
4-to-1 MUX

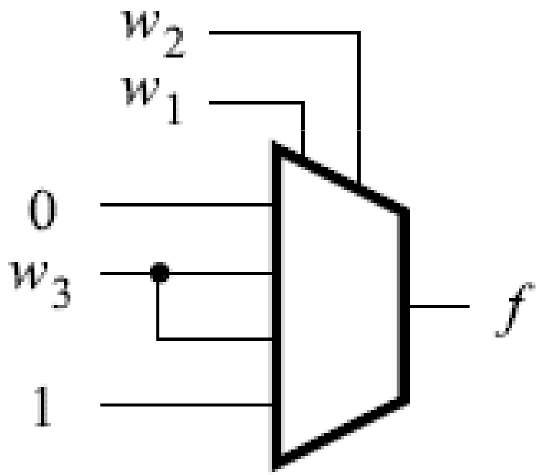
w_1	w_2	w_3	f	
0	0	0	0	w_3
0	0	1	1	
0	1	0	1	$\overline{w}_3$
0	1	1	0	
1	0	0	1	$\overline{w}_3$
1	0	1	0	
1	1	0	0	w_3
1	1	1	1	

3-input Majority Function

- Get 3 inputs and output 1 if # of 1's greater than # of 0's.

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	w_2	f
0	0	0
0	1	w_3
1	0	w_3
1	1	1



Shannon's Expansion

- Shannon's Expansion Theorem

$$f(w_1, w_2, \ldots, w_n) = w_1' f(0, w_2, \ldots, w_n) + w_1 f(1, w_2, \ldots, w_n)$$

$$= w_1' f_{w_1'} + w_1 f_{w_1}$$

$f_{w_1'}, f_{w_1}$: cofactors

- Example : 3-input majority function

$$f = w_1'w_2w_3 + w_1w_2'w_3 + w_1w_2w_3' + w_1w_2w_3 = w_1w_2 + w_2w_3 + w_1w_3$$

Can be rewritten as

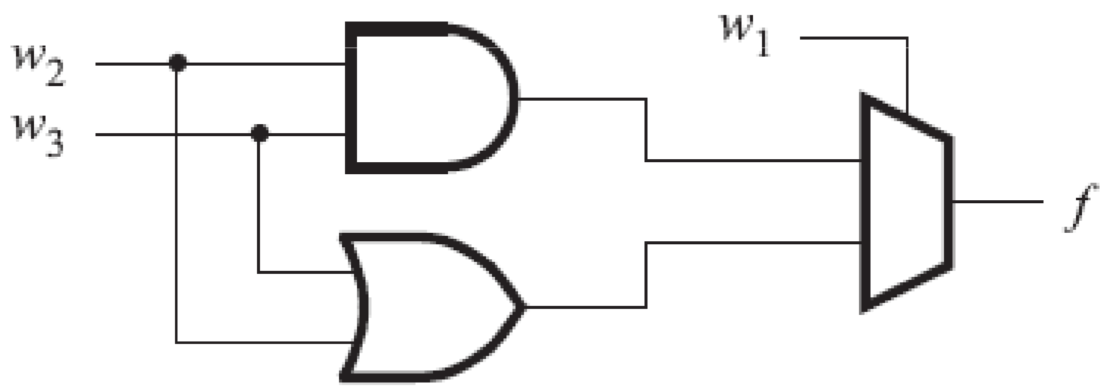
$$f = w_1w_2 + w_2w_3 + w_1w_3 = w_1'(w_2w_3) + w_1(w_2 + w_3)$$

Shannon's Expansion

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f
0	$w_2 w_3$
1	$w_2 + w_3$

Using
2-to-1 MUX



- 3-input XOR

$$f = w_1 \oplus w_2 \oplus w_3 = w_1'(w_2 \oplus w_3) + w_1(w_2 \oplus w_3)'$$

Shannon's Expansion

- In general : expand by w_i

$$f(w_1, w_2, \ldots, w_n) = w_i' f(w_1, w_2, \ldots, 0, \ldots, w_n) + w_i f(w_1, w_2, \ldots, 1, \ldots, w_n)$$
$$= w_i' f_{w_i'} + w_i f_{w_i}$$

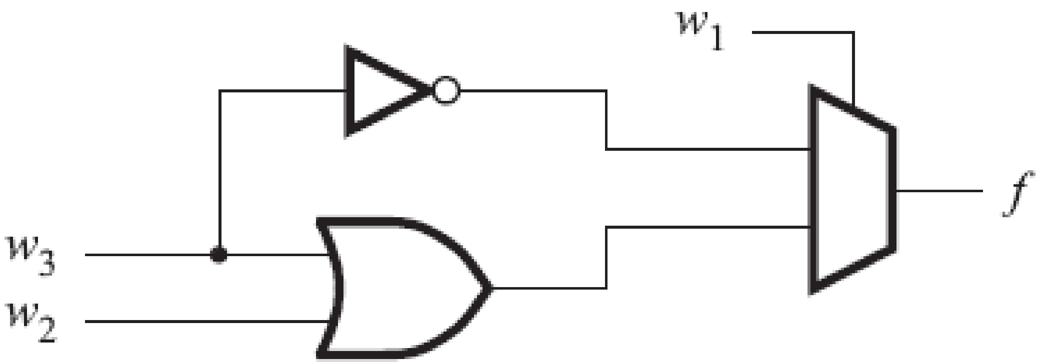
- 2-variable expansion:

$$f(w_1, w_2, \ldots, w_n) = w_1' w_2' f(0,0, w_3, \ldots, w_n) + w_1' w_2 f(0,1, w_3, \ldots, w_n)$$
$$+ w_1 w_2' f(1,0, w_3, \ldots, w_n) + w_1 w_2 f(1,1, w_3, \ldots, w_n)$$

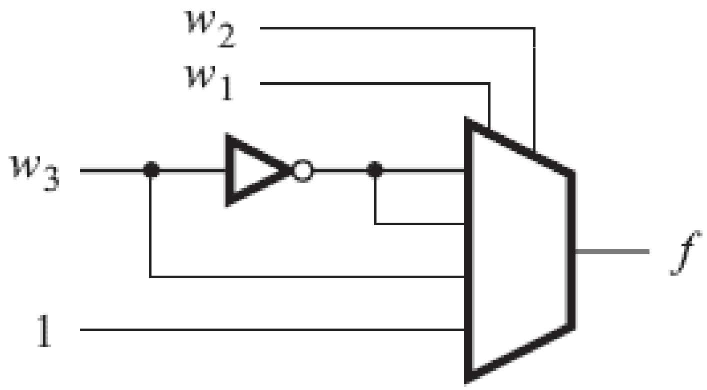
which can be implemented by a 4-to-1 MUX.

Example 1

$$f = w_1'w_3' + w_1w_2 + w_1w_3$$
$$= w_1'w_3' + w_1(w_2 + w_3)$$
$$= w_1'w_2'w_3' + w_1'w_2w_3' + w_1w_2'w_3 + w_1w_2(1)$$



Using
2-to-1 MUX



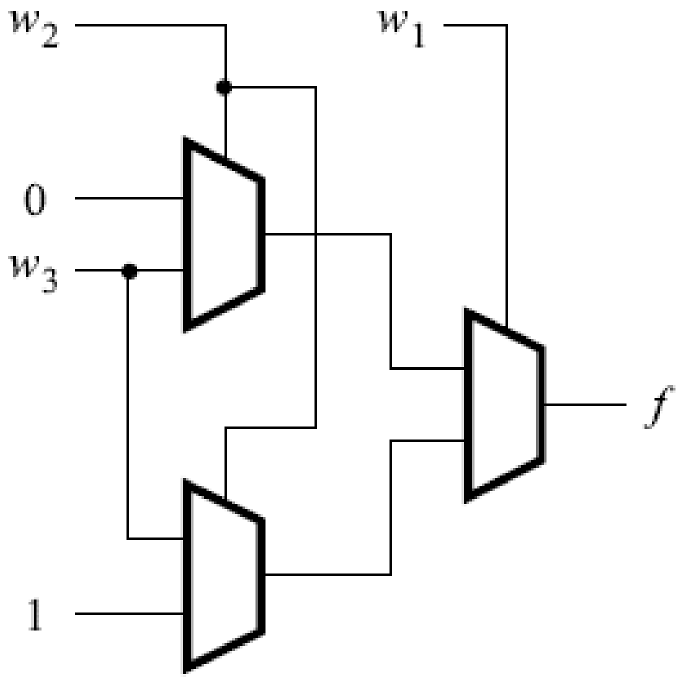
Using
4-to-1 MUX

3-input majority function

$$f = w_1 w_2 + w_2 w_3 + w_1 w_3 = w_1'(w_2 w_3) + w_1(w_2 + w_3)$$

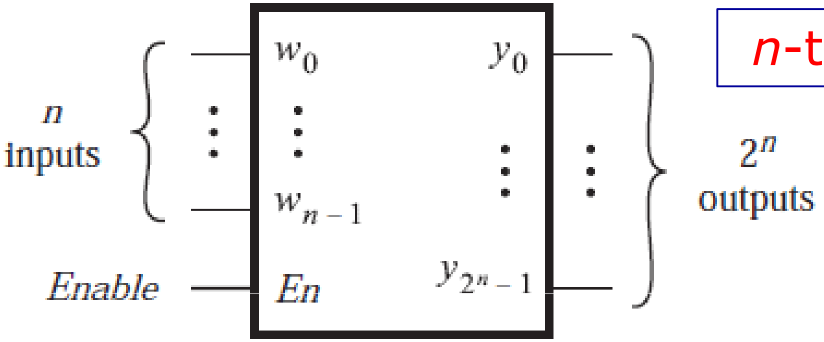
Let $g = w_2 w_3$, $h = w_2 + w_3$, then

$$g = w_2'(0) + w_2 w_3; h = w_2' w_3 + w_2(1)$$



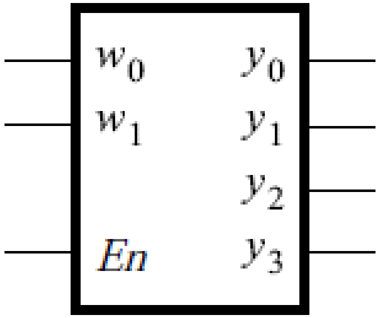
Decoder

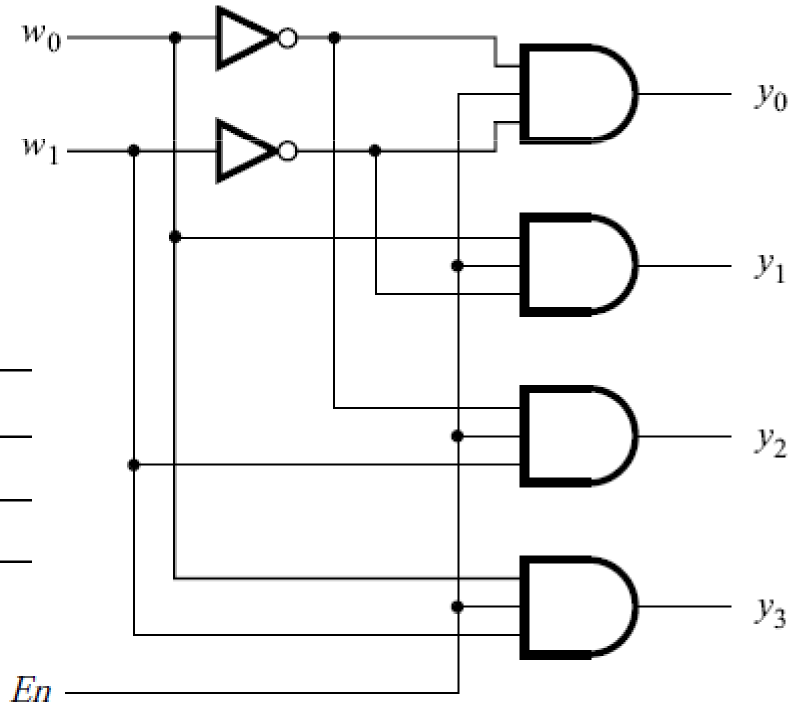
- Main function: decode "encoded" data.



n-to-2^n decoder

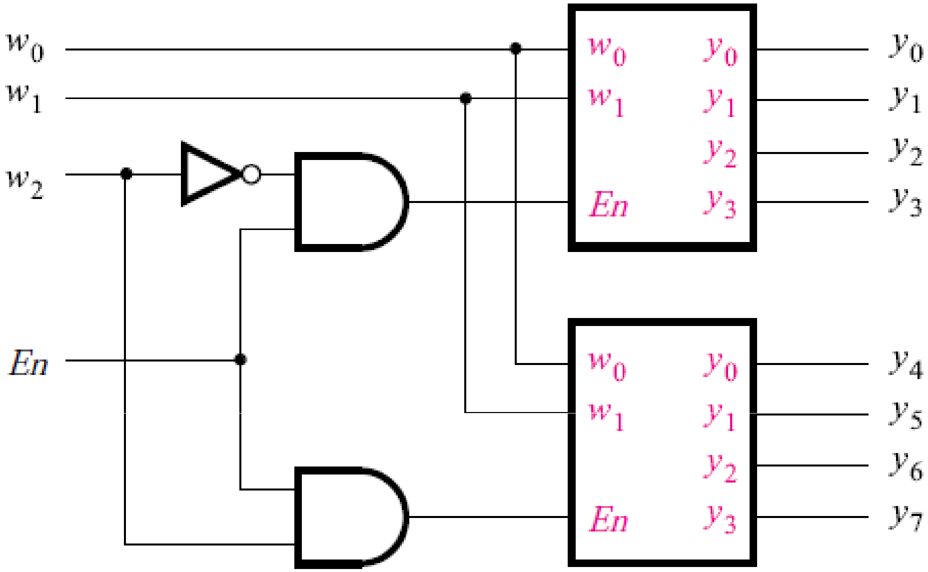
En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0



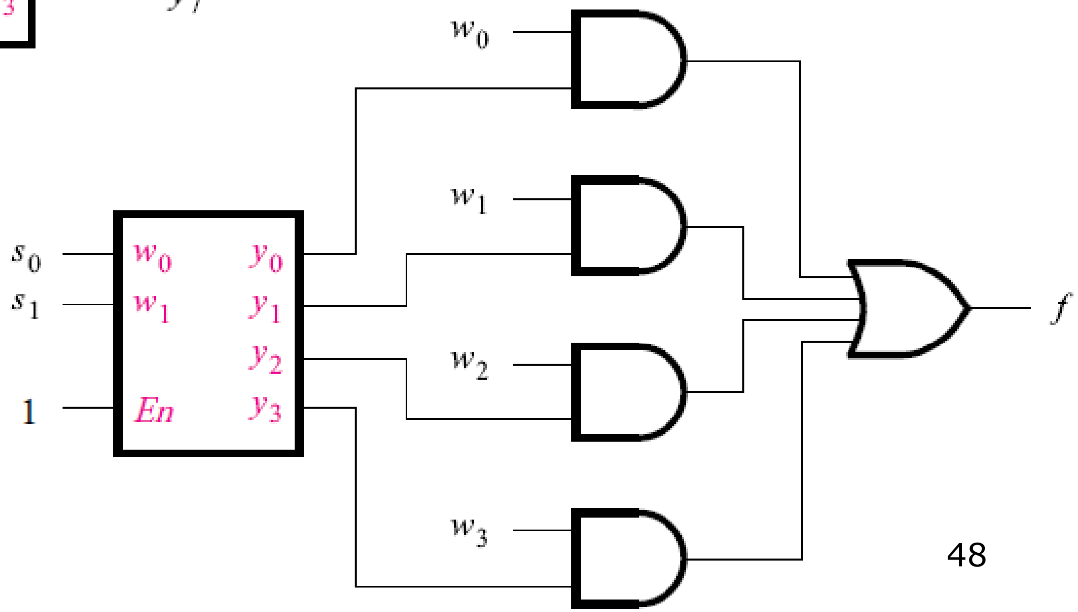


2-to-4 decoder

Decoder

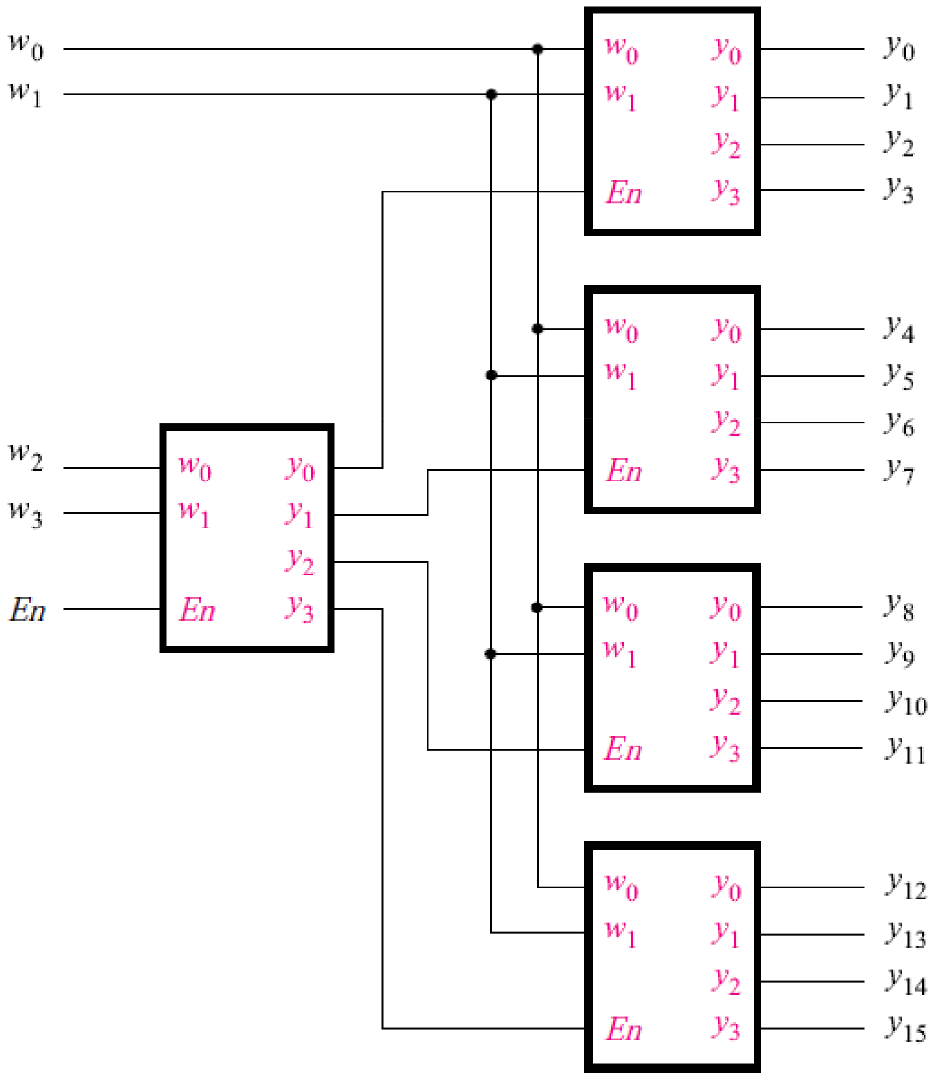


3-to-8 decoder using 2-to-4 decoder

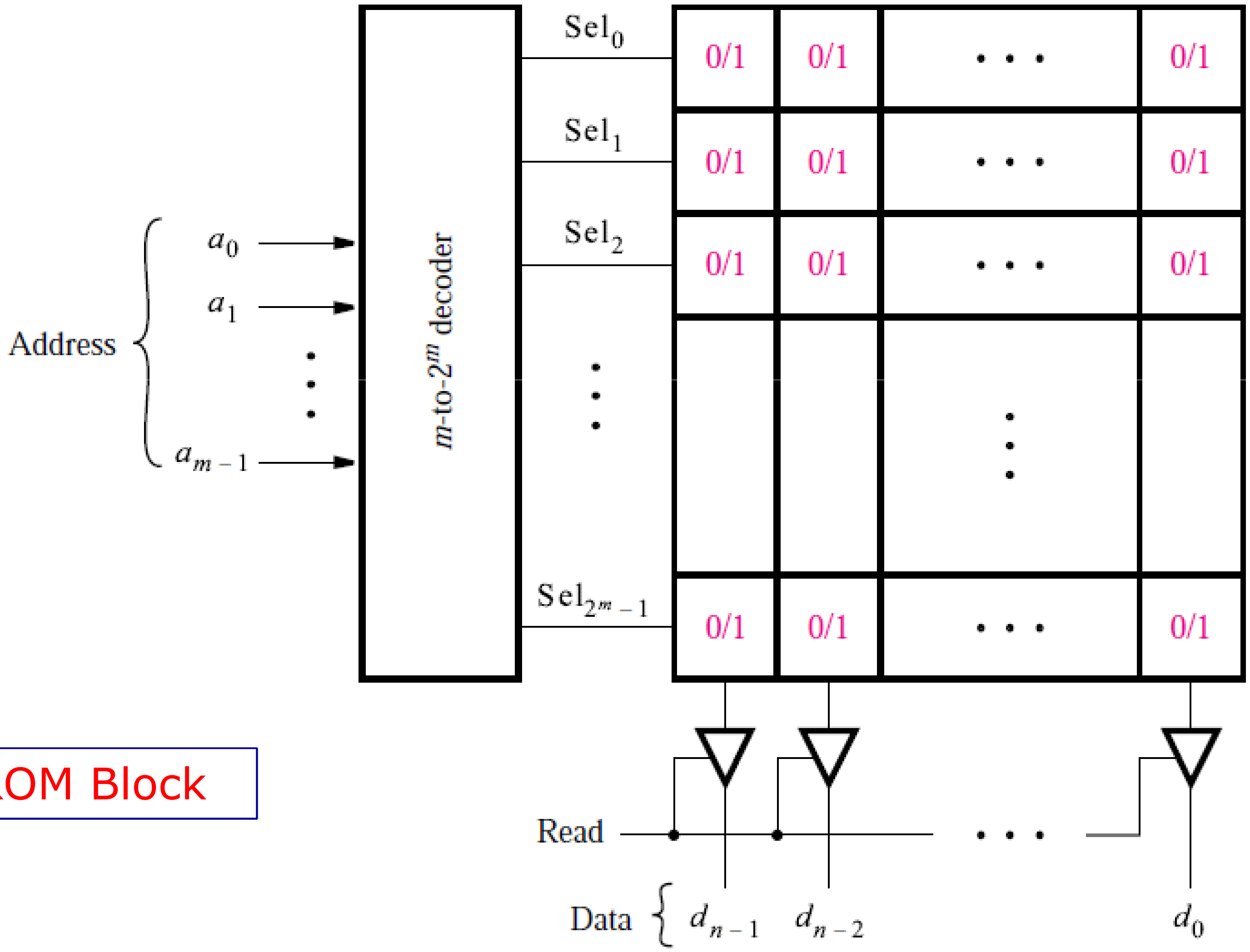


4-to-1 MUX using 2-to-4 decoder

4-to-16 Decoder

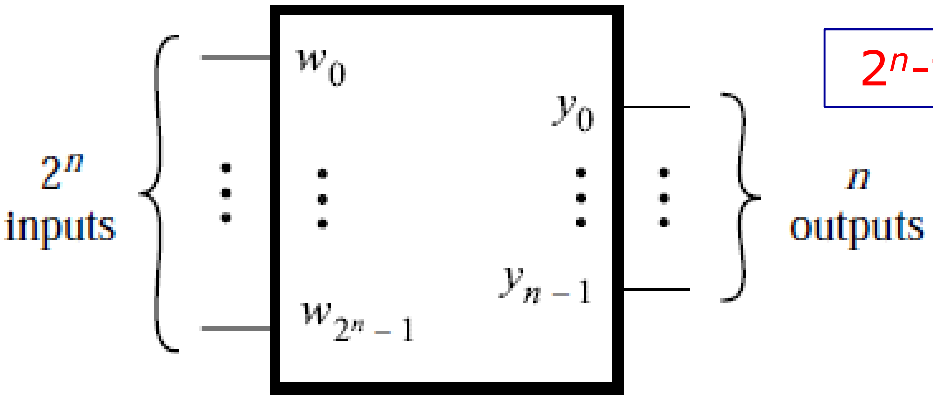


Demultiplexer (DEMUX)



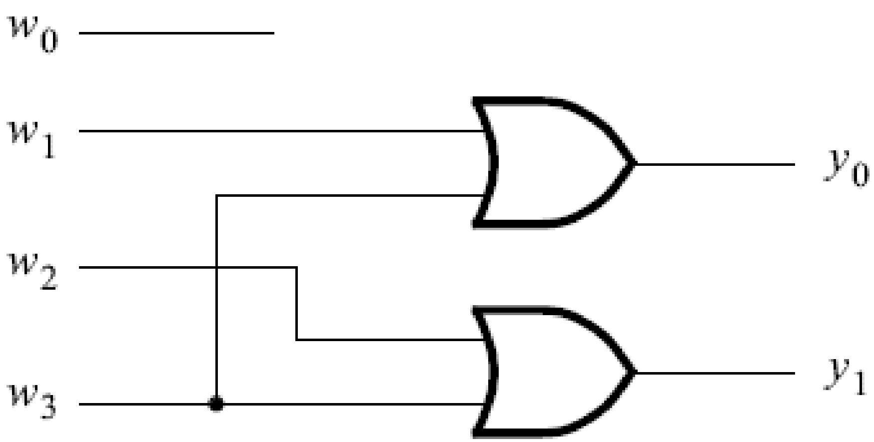
A $2^m \times m$ ROM Block

Encoder



2^n-to-n encoder

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

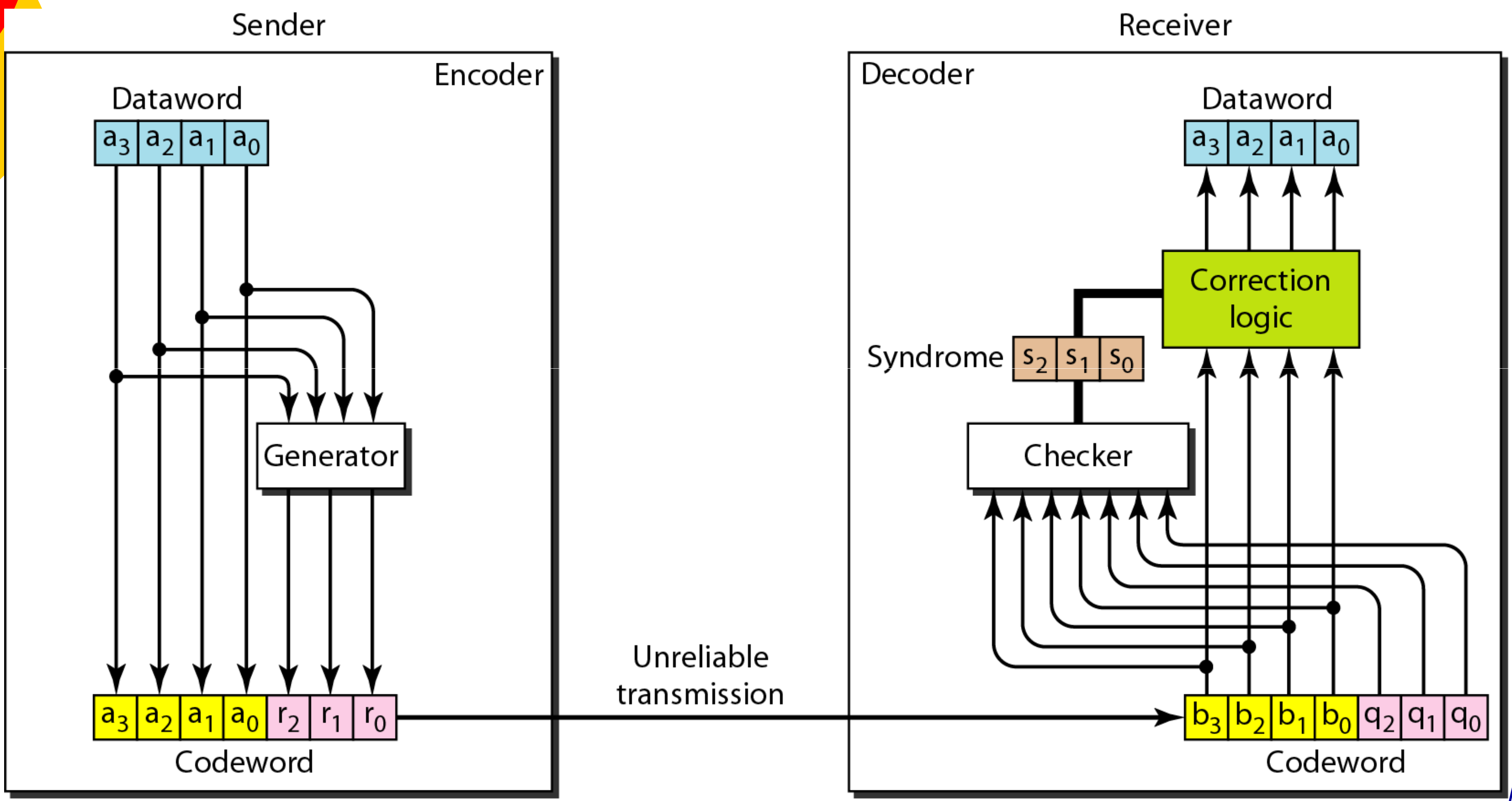


4-to-2 binary encoder

Hamming Code

- In “linear block code” family.
- Can correct 1-bit error or detect 2-bit error.
- Add parity bits to message bits.
- Typically use notation (n,k) Hamming code, which means n total bits, k message bits.
- Clearly there are (n-k) parity bits.

(7,4) Hamming Code



System Structure

Codewords

Datawords	Codewords	Datawords	Codewords
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

Codeword : $a_3a_2a_1a_0r_2r_1r_0$ with

$$r_0 = a_0 \oplus a_1 \oplus a_2 ; r_1 = a_1 \oplus a_2 \oplus a_3 ; r_2 = a_0 \oplus a_1 \oplus a_3$$

Syndrome

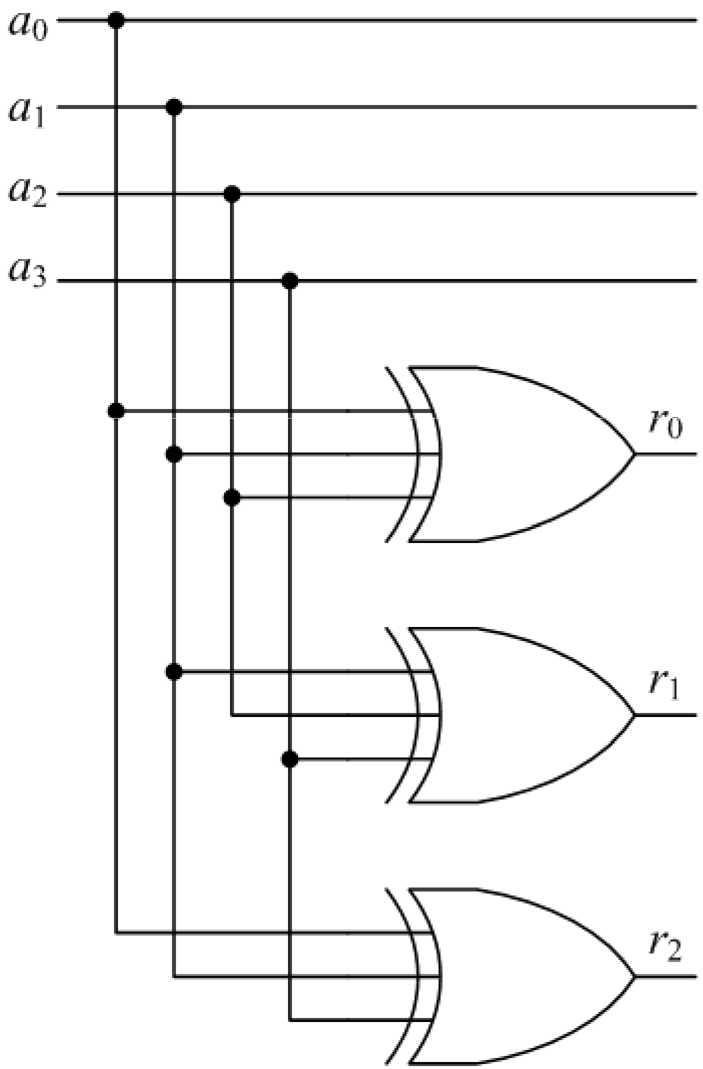
- Error pattern is given by "syndrome", i.e., $s_2 s_1 s_0$ (=**s**) where

$$s_0 = b_0 \oplus b_1 \oplus b_2 \oplus q_0 ; s_1 = b_1 \oplus b_2 \oplus b_3 \oplus q_1 ; s_2 = b_0 \oplus b_1 \oplus b_3 \oplus q_2$$

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	q_1	b_2	q_2	b_0	b_3	b_1

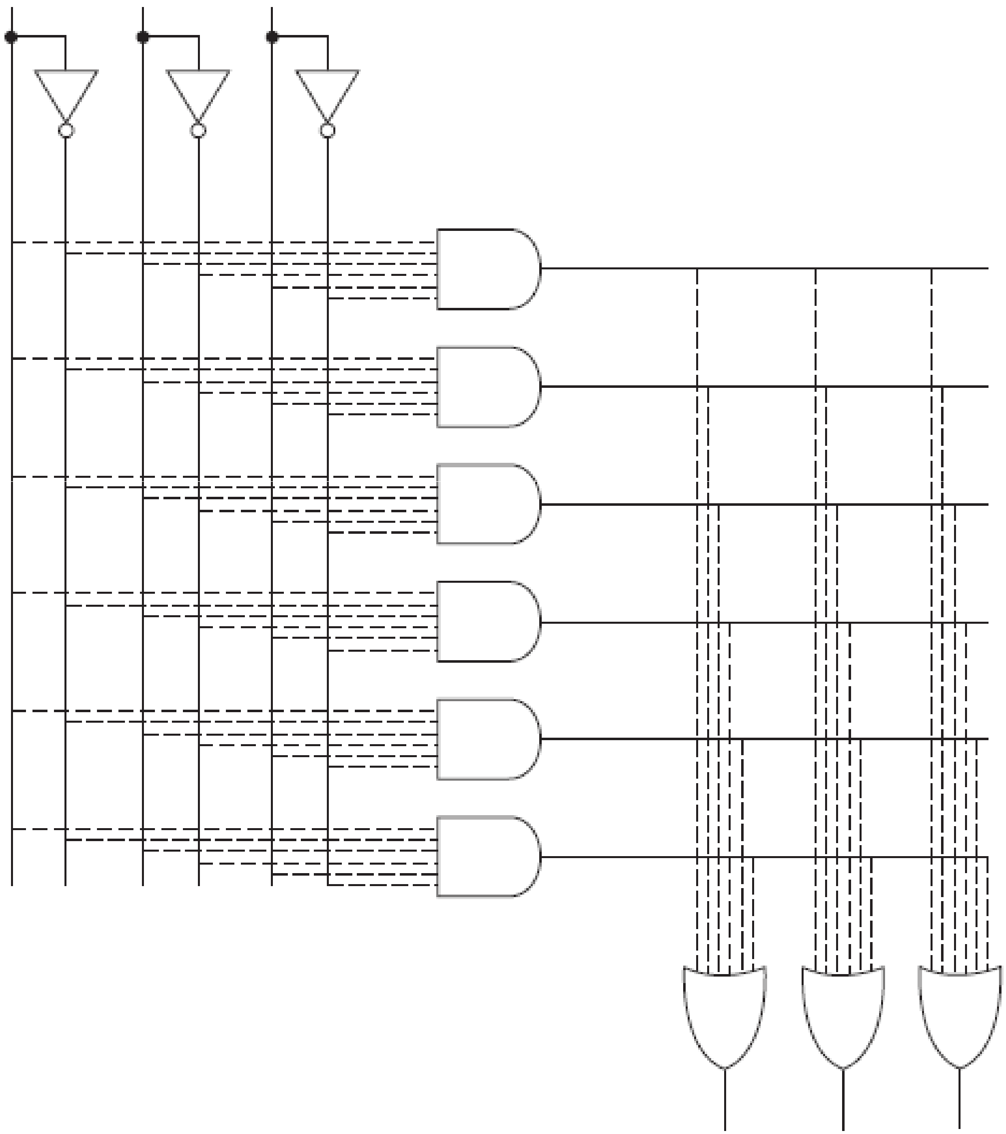
- Example : Send 0110100
  - Receive 0110100 -> **s** = 000 -> No error
  - Receive 0111100 -> **s** = 101 -> Error at b_0
  - Receive 0010100 -> **s** = 011 -> Error at b_2

(7,4) Hamming Encoder



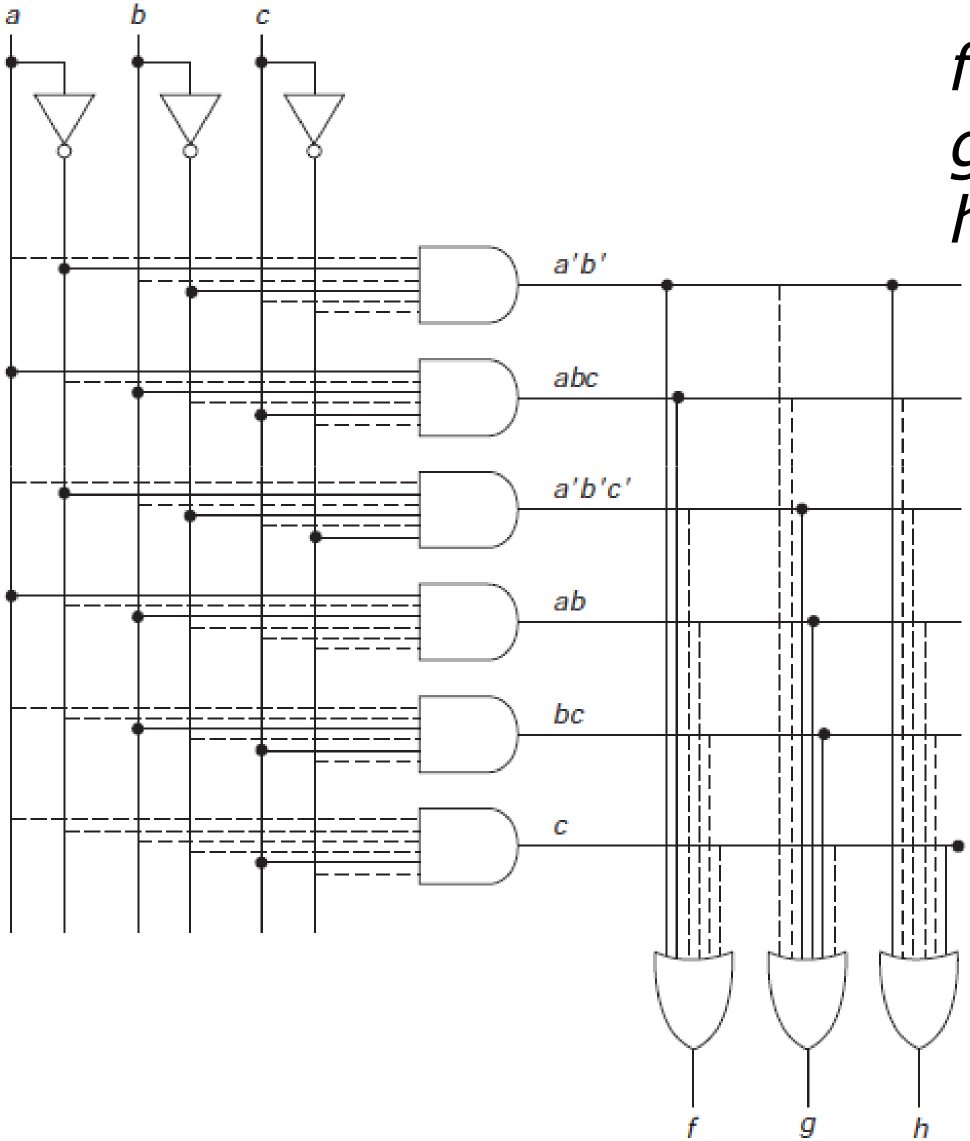
Exercise : Design the
(7,4) Hamming Decoder

Gate Arrays (Programmable Logic Device)



Basic Structure (AND-OR GA)

Example

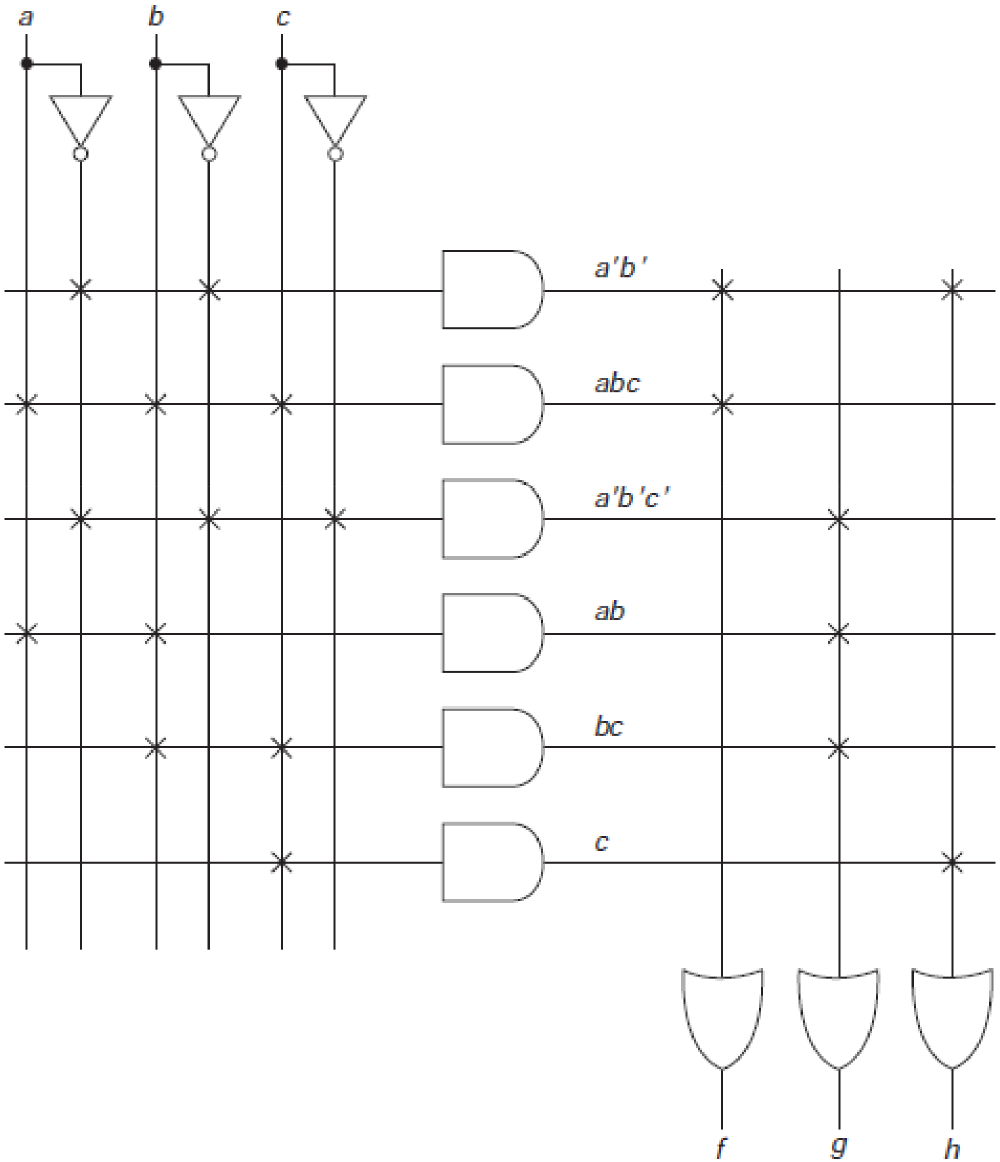


$f = a'b' + abc$

$g = a'b'c' + ab + bc$

$h = a'b' + c$

Simplified Diagram

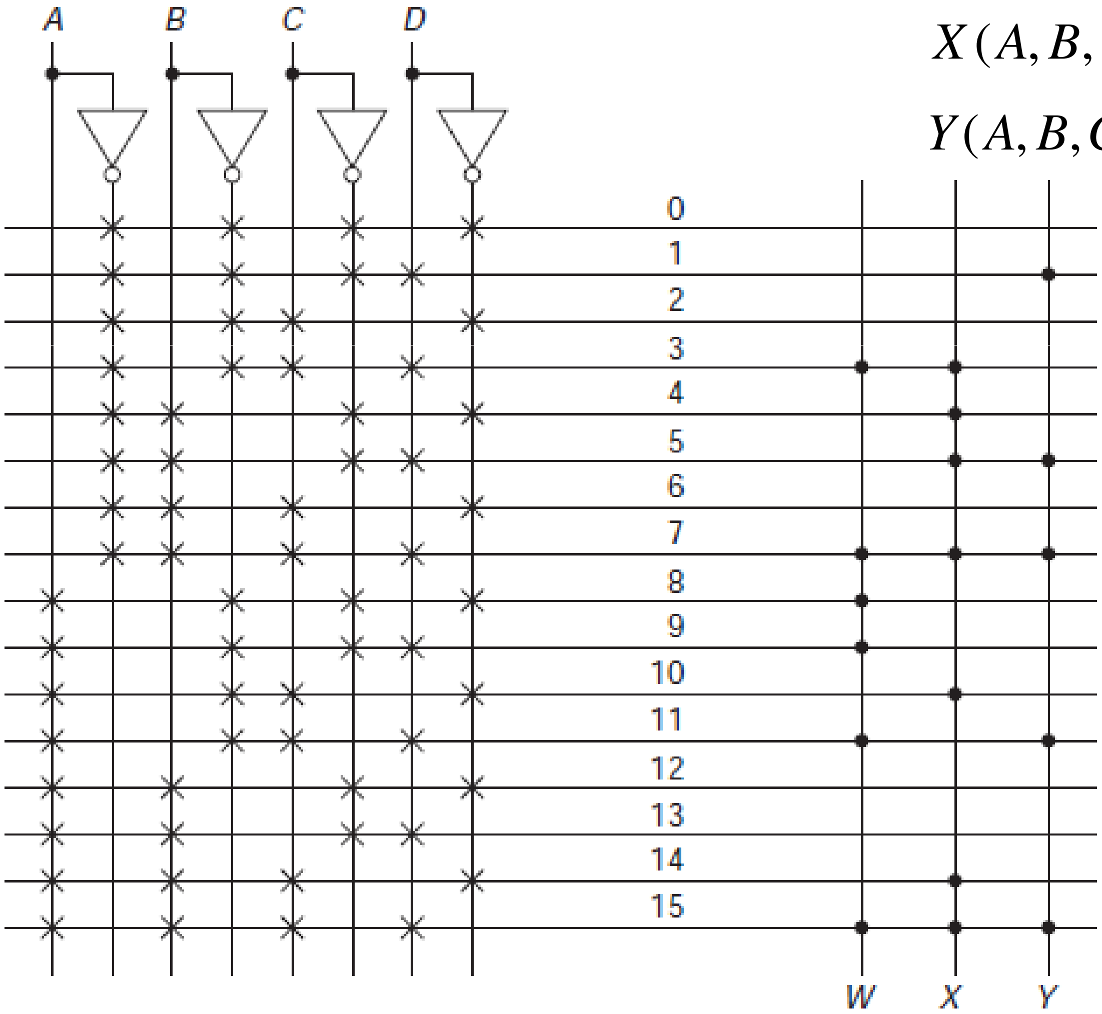


Using ROM

$$W(A,B,C,D)=\sum m(3,7,8,9,11,15)$$

$$X(A,B,C,D)=\sum m(3,4,5,7,10,14,15)$$

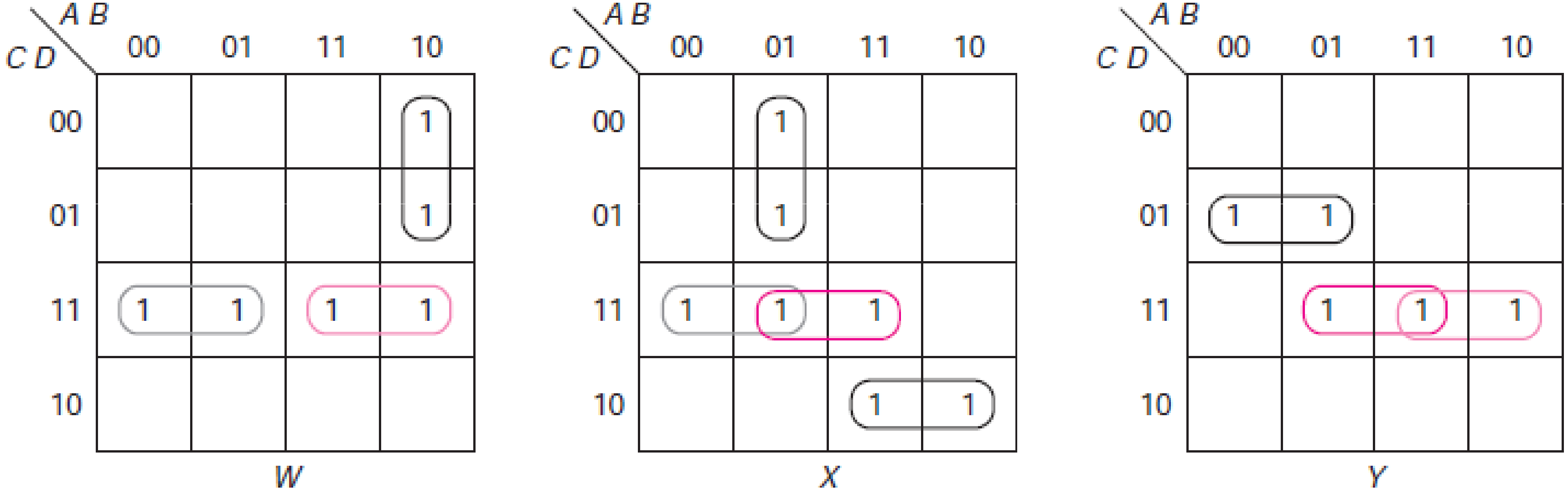
$$Y(A,B,C,D)=\sum m(1,5,7,11,15)$$



Using PLA

$$W(A,B,C,D)=\sum m(3,7,8,9,11,15);\ X(A,B,C,D)=\sum m(3,4,5,7,10,14,15)$$

$$Y(A,B,C,D)=\sum m(1,5,7,11,15)$$

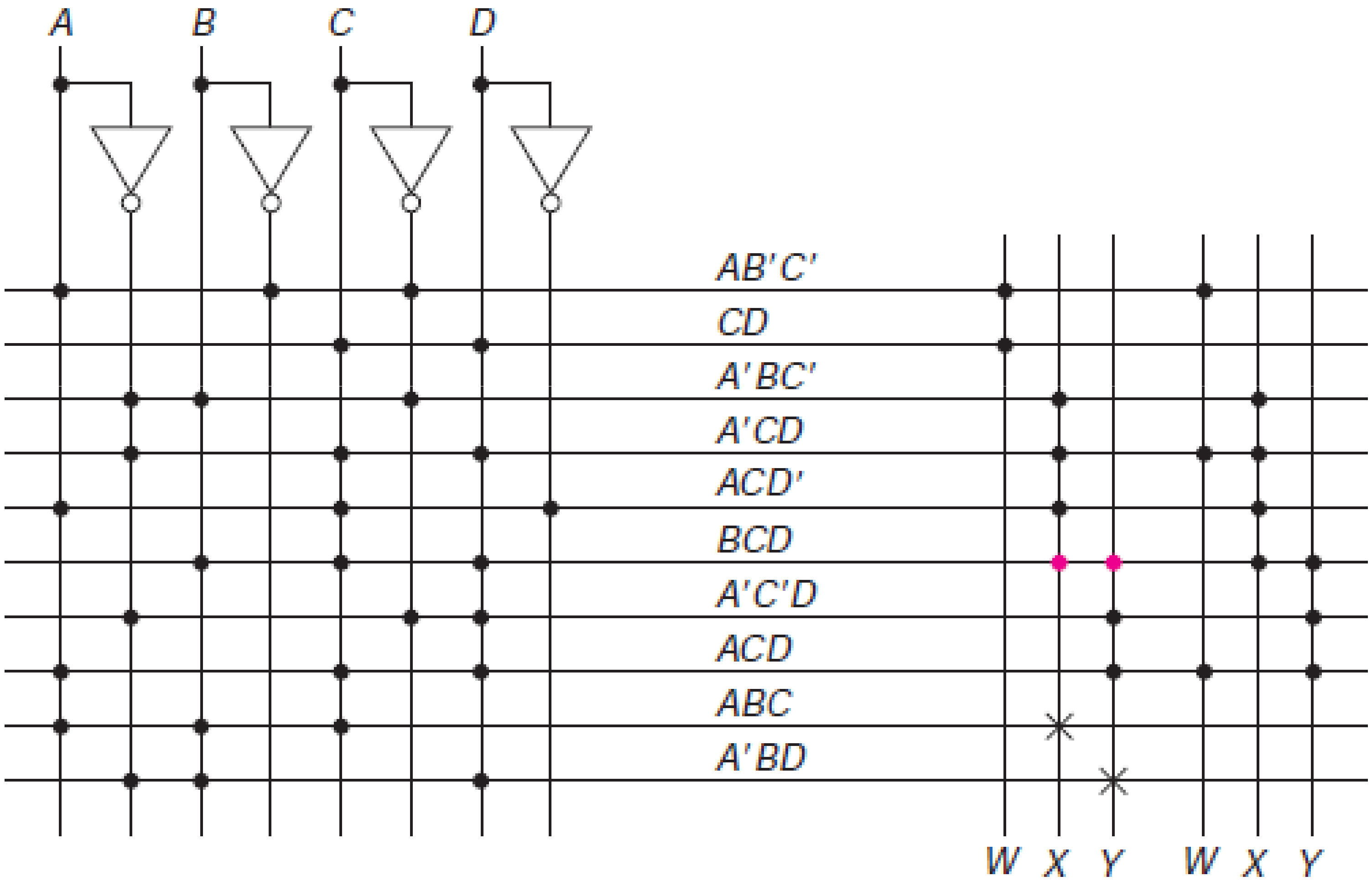


$$W = AB'C'+CD = AB'C'+A'CD+ACD$$

$$X = A'BC'+ACD'+A'CD+\{BCD \text{ or } ABC\}$$

$$Y = A'C'D+ACD+\{BCD \text{ or } A'BD\}$$

Using PLA (2)



Using Programmable Array Logic